IKS

INFORMATYKA KOMPUTERY SYSTEMY

CENA — 50 zł

Dodatek „Żołnierza Wolności" nr 61/1986 ISSN 0860-2794

Masz kłopoty z matematyką
— spójrz na str. 25

Grafika SHARPA
— str. 2 i 29

Uważni Czytelnicy z pewnością dostrzegają zachodzące na łamach naszego pisma zmiany; coraz więcej miejsca poświęcamy na prezentowanie programów na wszystkie najpopularniejsze w Polsce mikrokomputery. Staramy się, aby w większości były to programy użytkowe — takie, które mogą przydać się na co dzień. Chcemy, aby nasze publikacje zaspokajały nie tylko „pierwszy głód" informatycznej wiedzy, ale i satysfakcjonowały bardziej wymagających sympatyków mikrokomputerów.

Chroniczny brak literatury fachowej sprawia, iż użytkownicy Atari czy Commodore skazani są niemal na obcojęzyczne instrukcje. Dlatego też prezentujemy ciekawsze rozwiązanie programowe dotyczące poszczególnych typów mikrokomputerów.

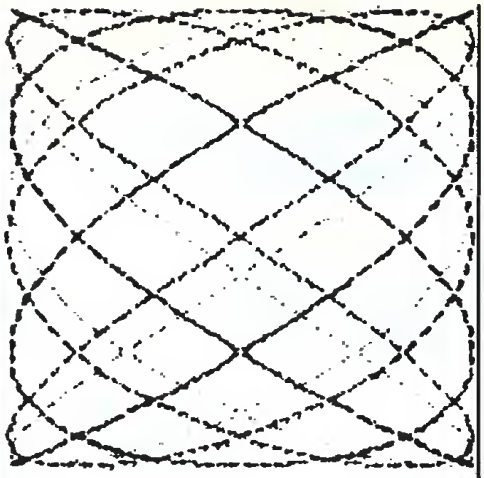
Liczymy oczywiście zawsze na pomoc Czytelników. Dobrych pomysłów z pewnością Wam nie brakuje. Nie brakowało ich nigdy — gorzej było z ich rozpowszechnianiem. Wiedza jest towarem najcenniejszym — szukajmy zatem talentów potrafiących robić właściwy użytek ze swych szarych komórek.

PROGRAM 87 — to hasło konkursu ogłoszonego przez Ogólnopolską Federację Klubów Komputerowych Młodych Mistrzów Techniki, któremu nasza redakcja wiernie sekunduje. Konkurs odbędzie się w trzech etapach: listopad — grudzień 1986, marzec — maj 1987 i wrzesień — listopad 1987 r. Programy oceniane będą w pięciu kategoriach: programy dydaktyczne, graficzne, gry komputerowe, programy sterujące robotami i automatami oraz programy rozwiązujące zagadnienia techniczne.

Federacja Klubów Komputerowych ostro wzięła się do roboty. Jeszcze w tym roku rozstrzygnięty będzie kolejny konkurs — tym razem na najlepiej pracujący klub komputerowy — a jest ich już ponad 200 (to tylko te, które należą do Federacji).

Komputery tanieją, ale jeszcze nieprędko osiągną odpowiednią cenę dla „popularnej pomocy szkolnej" — stąd też często właśnie klub komputerowy jest jedyną formą kontaktu z tym sprzętem. Być może problem zostanie rozwiązany, kiedy ruszy pełną parą produkcja elwrowskiego Juniora, ale tymczasem wiedza informatyczna młodzieży zależy często, na szczęście, od licznych zapaleńców, którzy uważają, że bez komputera nie ma przyszłości; skupiają wokół siebie podobnych pasjonatów, zdobywają środki i organizują klub komputerowy. Czasami o powodzeniu przedsięwzięcia decyduje przypadek. Tymczasem sukces powinien być regułą i do tego dążymy.

REDAKCJA

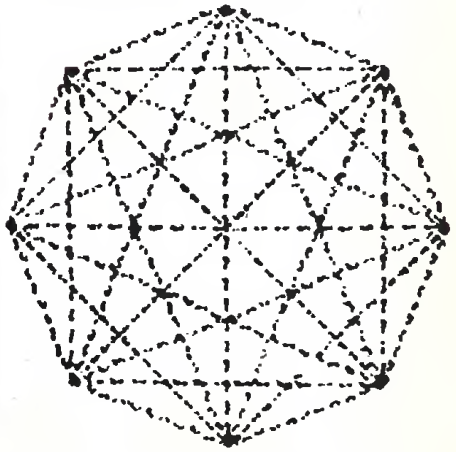


SHARP PC-1500
K R Z Y W E
LISSAJOUS

```
10:BEEP 10,100,10
   0
20:"A":CLEAR :
   USING :GRAPH :
   DEGREE :WAIT 0
30:INPUT "POZIOM
   X=";X:X=ABS X:
   X=INT X
40:INPUT "PION Y=
   ";Y:Y=ABS Y:Y=
   INT Y
50:IF (X+Y)=0GOTO
   10
60:INPUT "FAZA F=
   ";F
70:LINE -(0,0)-(1
   00,0),9:SORGN
80:R=75:T=0:X2=0:
   Y2=0
90:N=36*(X+Y):S=3
   60/N
100:X2=R*SIN (X*T)
   :Y2=R*SIN (Y*(
   T+F))
110:LINE -(0,0)-(X
   2,Y2),9
130:FOR K=0TO N
140:T=T+S:X1=X2:Y1
   =Y2
150:X2=R*SIN (X*T)
   :Y2=R*SIN (Y*(
   T+F))
160:LINE -(X1,Y1)-
   (X2,Y2),1
170:NEXT K
180:LINE -(X2,Y2)-
   (0,0),9
190:TEXT :END
```

SHARP PC-1500
KRYSZTAL

```
10:BEEP 10,100,10
   0
20:"A":CLEAR :
   USING :WAIT 0
30:INPUT "ILE BOK
   OW B=";B:B=ABS
   B:B=INT B
40:IF B<4GOTO 10
50:INPUT "WIELKOS
   C Q=";Q:Q=ABS
   Q:Q=INT Q
60:IF Q=0GOTO 10
70:A=2*π/B:DIM W(
   B,2)
80:FOR K=0TO (B-1
   )
90:M=K+1:W(M,1)=Q
   *COS (K*A)
100:W(M,2)=Q*SIN (
   K*A)
110:NEXT K
120:GRAPH :RLINE -
   (110,0),9
130:SORGN :RLINE -
   (Q,0),9
140:X2=0:Y2=0
150:FOR L=1TO B
160:X1=W(L,1):Y1=W
   (L,2)
170:LINE -(X2,Y2)-
   (X1,Y1),9
180:FOR K=LTO B
200:N=K+1:IF N>B
   GOTO 240
210:X2=W(N,1):Y2=W
   (N,2)
220:LINE -(X1,Y1)-
   (X2,Y2),1
230:LINE -(X2,Y2)-
   (X1,Y1),9
240:NEXT K:NEXT L
250:LINE -(X2,Y2)-
   (0,0),9
260:TEXT :END
```



Na początku był ENIAC ...

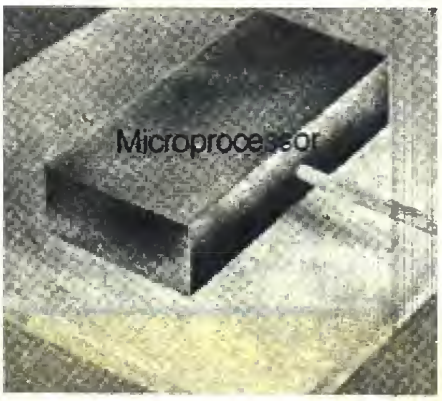
Wyprodukowanie pierwszego mikroprocesora stało się początkiem przewrotu, kojarzonego z drugą rewolucją przemysłową, zwaną rewolucją mikroprocesorową. Mimo dopiero piętnastoletniego istnienia, układ elektroniczny, nazwany w związku z technologią jego wykonania mikroprocesorem, nabrał historycznego znaczenia, znalazł wielostronne i niebanalne zastosowania.

Postęp w dziedzinie technologii elementów elektronicznych powodował kolejne kroki w rozwoju techniki komputerowej. Powstawały kolejne generacje komputerów. Oczywiście, warunkiem koniecznym ich rozwoju, poza doskonaleniem technologii, był rozwój koncepcji działania systemów obliczeniowych i oprogramowania.

Elementy elektroniczne wkroczyły do techniki obliczeniowej w 1946 r. Na Uniwersytecie Pensylwania skonstruowano wówczas pierwszy działający komputer ENIAC (Electronic Numerical Integrator and Computer). Ważył 30 ton (!), zużywał 150 kW energii elektrycznej (!), a zbudowany był z 18 000 lamp próżniowych i 1500 przekaźników elektromagnetycznych. Na wykonanie operacji dodawania dwóch liczb dziesięciocyfrowych ENIAC potrzebował 0,2 milisekundy (1 ms = 0,001 s), a na wykonanie operacji mnożenia 2,8 ms. Komputer posiadał: **procesor**, który tworzyły: **jednostka sterująca** i **arytmometr**; **pamięć wewnętrzną**, składającą się z 20 rejestrów mieszczących liczby dziesięciocyfrowe; **pamięć zewnętrzną**, na pliku kart dziurkowanych.

Komputer ENIAC działał w następujący sposób. Jednostka sterująca wprowadzała na wejście arytmometru daną liczbową, zapisaną w rejestrze pamięci lub na karcie dziurkowanej, a następnie wysterowywała arytmometr, co oznaczało wykonanie odpowiedniej operacji arytmetycznej. Wynik operacji wyprowadzany był do jednego z rejestrów. Kolejne działania procesora zaprogramowane były w tzw. tablicy punktów. Rozwiązanie problemu obliczeniowego oznaczało wprowadzenie właśnie tablicy punktów, których kolejność odpowiednich połączeń interpretowana była przez jednostkę sterującą oraz przygotowanie danych na kartach dziurkowanych.

Czasochłonność oraz niedogodność programowania oraz mała pojemność pamięci wewnętrznej, którymi to charakteryzował się ENIAC, „poprawione" zostały w koncepcji J. Eckerta i J. von Neumanna. Koncepcja ta ukazywała ogromną rolę **pamięci wewnętrznej (operacyjnej)** i jej wpływ na architekturę komputera, a polegała na tym, że jednostka sterująca mogła wykorzystywać, zapisane w tej samej pamięci, dane i **instrukcje programu (rozkazy)** — operujące na tych danych.



Architektura komputera, zgodnego z tą koncepcją, zawierać miała procesor, tor przesyłania danych i pamięć. Procesor tworzyły arytmometr i jednostka sterująca, która zawierała m.in. licznik rozkazów i rejestr rozkazów. Pamięć operacyjna zawierała, oprócz komórek pamięci, rejestr adresowy pamięci.

Cykl pobrania rozkazu przez komputer o takiej architekturze wyglądał następująco. W chwili początkowej licznik rozkazów ma wartość zerową. Zawartość licznika przesyłana jest do rejestru adresowego pamięci. Po czasie, zwanym czasem dostępu do pamięci, zawartość komórki o adresie zerowym przesyłana jest do rejestru rozkazów. Rozkaz po pobraniu jest interpretowany przez procesor, a następnie wykonywany.

Rozkaz procesora składa się z **części operacyjnej**, wskazującej, jaką operację ma wykonać procesor oraz z **części adresowej**, wskazującej adres komórki pamięci (adresy komórek pamięci), w których, zawarte są argumenty tej operacji.

Do momentu sformułowania koncepcji Eckerta i von Neumanna do budowy komputerów wykorzystywano **próżniowe lampy elektronowe** z trzema elektrodami: katodą, anodą i siatką, lampy zwane triodami, o zdolnościach wzmacniających. Jako elementy logiki wykorzystywano **przekaźniki elektromagnetyczne**. Pamięć zewnętrzną stanowiły wspomniane wcześniej **karty dziurkowane (perforowane)**, na których informacje kodowane są za pomocą tzw. kodu Holleritha. Konieczne stało się konstrukcyjne rozwiązanie „dużej" pamięci operacyjnej, o pojemności kilku tysięcy komórek.

W komputerze EDVAC, uruchomionym w 1950 r., do budowy pamięci wykorzystano **rtęciowe linie opóźniające**. Odznaczały się dużą szybkością działania, ale małą pojemnością i były szkodliwe dla ludzi, zostały więc szybko wyparte.

Rozpoczęła się era panowania techniki komputerowej. Opracowywano nowe technologie i konstrukcje, poszukiwano nowych rozwiązań funkcjonalnych.

Powstały trzy podstawowe typy **pamięci elektromechanicznych:** pamięć bębnowa, pamięć taśmowa i pamięć dyskowa.

W **pamięci bębnowej** głowice magnetyczne zapisują (przez lokalne namagnesowanie) lub odczytują informacje z wirującego bębna, na który naniesiony jest materiał magnetyczny. Z uwagi na konieczność ustawienia głowic na odpowiedni obszar bębna pamięć bębnowa jest powolna, ale ma ważną zaletę — dużą pojemność. Czas dostępu do informacji zazwyczaj wynosi kilkadziesiąt ms. Pamięć umożliwia zapisanie na jednym bębnie do miliona bitów (bit — binary digit — najmniejsza jednostka do reprezentacji danych dwuwartościowych, czyli zer lub jedynek liczbowego systemu dwójkowego).

dokończenie na str. 4

dokończenie ze str. 3

Pamięci na taśmie magnetycznej stosowane są do dziś, ze względu na znaczną pojemność, mimo stosunkowo długiego czasu dostępu do potrzebnej informacji.

Pamięci dyskowe, wykorzystujące lokalny zapis magnetyczny, znalazły trwałe zastosowanie w komputerach, ze względu na dużą pojemność, rzędu miliardów bitów w konstrukcjach wielodyskowych. Czas dostępu do informacji wynosi kilkanaście ms.

Wymienione pamięci elektromechaniczne stosowane są jako masowe pamięci zewnętrzne.

Bardzo duży wpływ na rozwój techniki komputerowej wywarły **rdzeniowe pamięci ferrytowe.** Charakteryzowały się one dużą szybkością odczytu, rzędu kilku mikrosekund (1 μs = 0,000001 s). Były one wykorzystywane jako pamięci wewnętrzne, wyparte zostały dopiero przez **pamięci półprzewodnikowe.**

Odkrycie w 1948 r. przez W. Shokleya, J. Bardeena i W. Brattaina nowego, półprzewodnikowego elementu elektronicznego, **tranzystora,** który zastąpił jedyny dotąd element wzmacniający — lampę elektronową, to początek ery tranzystorowej w elektronice i początek II generacji techniki komputerowej.

Zanim przejdziemy dalej, kilka zdań o innych konstrukcjach komputerowych I generacji.

W 1951 r. firma UNIVAC polecała swój komputer UNIVAC—1, zbudowany z 5000 lamp i z 112 rtęciowych linii opóźniających, mieszczących 1000 liczb dwunastocyfrowych. Wykonywał on 45 rozkazów z wyróżnioną częścią operacyjną i częścią adresową (trzycyfrową). Do komunikacji operatora z systemem po raz pierwszy wykorzystano elektryczną maszynę z klawiaturą, natomiast do zapamiętywania dużych zbiorów danych wykorzystywano taśmę magnetyczną. Operację dodawania dwóch liczb dwunastocyfrowych UNIVAC-1 wykonywał w 0,5 ms, operację mnożenia w 2,5 ms. Już w rok później w komputerze UNIVAC 1103 jako pamięć operacyjną zastosowano rdzenie magnetyczne. Zwiększyło to pięćdziesięciokrotnie szybkość UNIVAC 1103 w stosunku do UNIVAC-1.

W 1952 r. firma IBM weszła na rynek komputerowy z udaną (nie pierwszą) konstrukcją, komputerem 701. Zawierał on 4000 lamp i 12 000 diod germanowych. Jako pamięć operacyjną zastosowano 72 pamiętające lampy katodowe, o czasie dostępu 12 μs. W komputerze zastosowano kilka urządzeń wejścia — wyjścia systemu: czytnik i dziurkarkę kart, drukarkę o szybkości 150 wierszy/minutę, taśmę magnetyczną o gęstości zapisu 100 bitów/cal. IBM 701 wykonywał operacje na danych 36-bitowych, operacja dodania dwóch liczb trwała 62,5 μs, operacja mnożenia 500 μs. Kolejne modele firmy IBM 702, 704, 705 miały pamięci rdzeniowe. W następnych modelach RAMAC 305 i 650 jako masową pamięć zewnętrzną zastosowano pamięci dyskowe, panele złożone z 50 dysków. Na każdym z dysków znajdowało się 100 ścieżek, o pojemności 10 000 znaków każda. Pamięć wyróżniała się więc olbrzymią pojemnością.

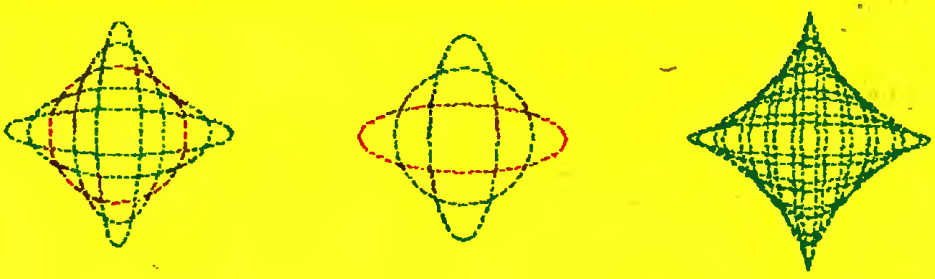
Przedstawiony wyżej etap rozwoju techniki komputerowej charakteryzował się pogonią za parametrami technicznymi. Niewiele uwagi poświęcono systemom operacyjnym i językom programowania, sterowanie pracą komputera oznaczało żmudne pisanie programów w języku procesora. Jako pierwsza tymi problemami zajęła się firma UNIVAC. Komputer UNIVAC 1103 wyposażony został w **interpreter**, program tłumaczący na język procesora kolejne wiersze programu. Po przetłumaczeniu danego wiersza procesor komputera wykonywał odpowiednie rozkazy, następnie czekał na przetłumaczenie kolejnego wiersza programu.

W firmie UNIVAC uruchomiono również program **translatora,** tłumaczącego na język procesora od razu cały program. Po przetłumaczeniu procesor wykonywał wszystkie rozkazy. Taka metoda nazywana jest **kompilacją**, a program **kompilatorem.**

Bardzo ważną sprawą jest organizacja pracy systemu komputerowego czyli współdziałania wszystkich elementów. Zbiór programów obejmujących te zagadnienia nazywa się **systemem operacyjnym.**

Kolejnym problemem są **języki programowania,** umożliwiające pisanie programów w języku zbliżonym do ludzkiego, możliwie dalekim od języka procesora. Dany język jest zazwyczaj przeznaczony do rozwiązywania problemów z pewnej dziedziny. Wśród języków programowania współcześnie „panujących" najważniejsze to: **ALGOL**, do opisu algorytmów obliczeń; **FORTRAN**, do zastosowań naukowo-technicznych; **COBOL,** do przetwarzania dużej liczby danych; **BASIC**, do rozwiązywania problemów naukowo-technicznych; **C,** do prac nad systemami operacyjnymi; **PASCAL,** do operacji na złożonych strukturach danych; **LOGO,** do nauki podstaw informatyki; **PROLOG,** do programowania logicznego i oprogramowania robotów.

JANUSZ MILLER
SHARP PC-1500
E L I P S Y



```
10:BEEP 10,100,100
20:"A":CLEAR :USING :RADIAN
30:INPUT "LICZBA ELIPS ? N=";N:N=
   ABS N:N=INT N:IF N<360TO 10
40:INPUT "A+B=M ? M=";M:M=ABS M:IF
   M=80R M>9860TO 10
50:F=π/20:D=M/(N+1):S=π/20
70:FOR K=1TO N
80:C=C+1:IF C>8LET C=C-3
90:A=M-K*D:B=K*D:GRAPH
100:COLOR C:LINE -(0,0)-(110,0),9:
    SORGN
110:LINE -(0,0)-(A,0),9:X2=A:Y2=0:T=
    0
120:FOR L=0TO 39
130:X1=X2:Y1=Y2:T=T+S
140:X2=A*COS T:Y2=B*SIN T
150:LINE -(X1,Y1)-(X2,Y2),1
160:NEXT L
170:LINE -(X2,Y2)-(0,0),9
180:TEXT :NEXT K
190:END
```

Wspomniane problemy oprogramowania większe odzwierciedlenie znalazły w komputerze UNIVAC III.

Opanowanie technologii tranzystorów, zwłaszcza tranzystorów bipolarnych, to podstawa konstrukcji komputerów II generacji. Tranzystory unipolarne, inaczej polowe, konstruowane na strukturze MOS (Metal — Oxide — Semiconductor), wykorzystywane były tylko w **układach peryferyjnych** systemów komputerowych. Technologia MOS nabrała znaczenia dopiero w latach siedemdziesiątych.

W latach sześćdziesiątych poczyniono kolejny krok, na jednym kryształku półprzewodnika umieszczono więcej niż jeden element. Zaczęto konstruować **układy scalone**, zawierające wiele elementów — tranzystorów, diod, rezystorów itp.

Układy scalone, w zależności od liczby elementów w strukturze półprzewodnika, można podzielić m.in. na: małej skali integracji SSI (Small Scale of Integration) — zawierające do 100 tranzystorów w swojej strukturze i średniej skali integracji MSI (Medium Scale of Integration) — zawierające do 1000 tranzystorów.

Układy scalone SSI i MSI posłużyły do konstrukcji komputerów III generacji. Wymienione układy scalone wykonywane były głównie technologiami wykorzystującymi tranzystory bipolarne. Z technologii tych największe znaczenie osiągnęła TTL (Transistor — Transistor Logic), która króluje na rynku elektronicznym do dziś.

W związku z wprowadzeniem **elementów półprzewodnikowych**, zapewniających wzrost szybkości działania i niezawodność, zmniejszenie poboru mocy oraz spadek kosztów wykonania oraz w związku z pracami nad oprogramowaniem, sprzęt komputerowy stał się bardziej dostępny, powszechniejszy. Bardzo popularnym był komputer IBM 1360 oraz najszybszy do 1979 r. komputer ILLIAC IV, wykonujący 200 mln rozkazów/sekundę(!).

Na rynku komputerowym zaczęły panować dwie tendencje. Pierwsza z nich to upowszechnienie szybkich i dużych komputerów. Druga to miniaturyzacja systemów komputerowych, przy zachowaniu parametrów technicznych dużych systemów.

W 1970 r. firma Digital Equipment Corporation wprowadziła PDP-11, komputer o bardzo oryginalnej architekturze. Wszystkie elementy tego komputera: procesor, pamięć i urządzenia wejścia-wyjścia, dołączone były do jednej wspólnej szyny.

W tym czasie opanowano technologie wytwarzania układów scalonych dużej skali integracji LSI (Large Scale of Integration), zawierających do 10 000 tranzystorów w jednej strukturze półprzewodnikowej. Rozpoczęła się era dominacji **technologii MOS**, zwłaszcza PMOS, NMOS, CMOS a ostatnio CHMOS. W 1971 r. zaprojektowano i wykonano pierwszy **mikroprocesor**. Rozpoczęła się IV generacja komputerów. Układy LSI to na pewno największe osiągnięcie elektroniki, od czasu wynalezienia tranzystora.

Lata osiemdziesiąte to nowe układy scalone, VLSI (Very Large Scale of Integration) o olbrzymich możliwościach. Przykładowo koprocesor arytmetyczny firmy MOTOROLA MC 68881, wykonany w technologii CHMOS, przeznaczony do współpracy z 32-bitowym mikroprocesorem M 68020, zawiera 155 000 tranzystorów w strukturze o wymiarach 27 na 33 mm.

Tak w skrócie przeszliśmy do komputerów nam współczesnych. Reasumując — komputer to: architektura i jej elementy składowe, czyli hardware; system operacyjny i języki programowania, interpretery i kompilatory, czyli software. W następnych numerach zajmiemy się tym pierwszym, czyli elementami elektronicznymi, z których zbudowane są komputery.

Jacek WOJTALA

PROGRAM 27 SPECTRUM

POSZERZANIE TABLICY ZMIENNYCH — Spectrum 48K

Jak 2-wymiarową tablicę zmiennych tekstowych nS, w której przechowujesz np. pewną liczbę nazwisk, powiększyć o kilka zmiennych? Program wczyta zmienną (linia 9900), ustali jej wymiary (linie 9902—9910), zapyta o liczbę dodatkowych nazwisk (linie 9912—9916), powiększy odpowiednio tablicę (linie 9918—9932) oraz nagra nową tablicę na taśmę (linia 9936). Aby przyspieszyć te operacje nie używamy pętli FOR—NEXT. Poniższą sekwencję możesz stosować jako część własnych, większych baz danych.

```
9900 LOAD "nazwiska" DATA n$()
9902 LET n$(1)=n$(1)
9904 RANDOMIZE FN p(23629)
9906 LET wymiar1= FN p( FN p(23670)-4)
9908 LET wymiar2= FN p( FN p(23670)-2)
9910 INPUT "ile nazwisk extra? "¡ile
9912 IF ile<0 OR INT ile<>ile THEN GOTO
 9910
9914 IF NOT ile THEN STOP
9916 DIM r$(wymiar1,wymiar2)
9918 LET a=1
9920 IF a<=wymiar1 THEN LET r$(a)=n$(a)
: LET a=a+1: GOTO 9920
9922 DIM n$(wymiar1+ile,wymiar2)
9924 LET a=1
9926 IF a<=wymiar1 THEN LET n$(a)=r$(a)
: LET a=a+1: GOTO 9926
9928 LET a=1
9930 IF a<=ile THEN INPUT "nazwisko "¡(
wymiar1+a), LINE n$(wymiar1+a): LET a=a
+1: GOTO 9930
9932 DIM r$(1)
9934 SAVE "nazwiska" DATA n$()
9940 DEF FN p(a)=PEEK a+256* PEEK (a+1)
```

PROGRAM 28

SPECTRUM

„Kto zabierze ostatnie jabłko ten wygra!" **Twoim przeciwnikiem jest SPECTRUM. Znajdujecie jabłka i bierzecie je na zmianę. Ostatnie jabłko jest najsmaczniejsze. Czy komputer zawsze musi je zabrać? Spróbuj odkryć „system" i zacznij wygrywać. W trakcie zastanawiania się słuchasz muzyki aleatorycznej (przypadkowej — linia 260). Jeżeli chcesz zawsze zaczynać grę jako pierwszy skasuj linię 230.**

Powodzenia i... smacznego!

```
  0>REM Jerzy Wozniak
  10 BORDER 0: PAPER 0: INK 7
  20 CLS
  30 PRINT "  KTO ZABIERZE OSTAT
NIE JAB£KO"
  40 PRINT
  50 PRINT "          TEN WYGRA !"
  60 PRINT
  70 PRINT "  WOLNO BRAC 1,2 LUB
 3 JAB£KA."
  80 PRINT "_____________________
_____"
  90
 100 REM LOSOWANIE JAB£EK
 110 LET p=INT (RND*14)+11
 120
 130 PRINT AT 8,18;"JAB£EK ";p
 140
 150 PRINT AT 11,0;"............
............"( TO p)
 160
 170 PRINT AT 12,0;"@@@@@@@@@@@@@
@@@@@@@@@@@@@"( TO p)
 180
 190 REM MIGACZ
 200 POKE 22912,143
 210
 220 REM LOSOWANIE=KTO ZACZYNA
 230 IF RND<0.5 THEN GO TO 360
 240 PRINT AT 16,0;
     "ILE JAB£EK ZABIERASZ ? ";
 250 LET r=CODE INKEY$-48
 260 PAUSE 12: BEEP 0.01,RND*30
 270 IF r<1 OR r>3
     THEN  GO TO 250
 280 PRINT r
 290 PRINT AT 18,23;" "
 300 LET p=p-r
 310 PRINT AT 9,2;
     "ZOSTA£O ";p;" "
 320 GO SUB 540
 330 IF p<1 THEN GO TO 510
 340
 350 REM KOMPUTER MYSLI
 360 LET n=INT (p/4)*4
 370 IF p=n THEN LET
     r=INT (RND*3+1)
 380 IF p<>n THEN LET r=p-n
 390 LET p=p-r
 400 PAUSE 60
 410 BEEP 1,10*r
 420 PRINT AT 18,5;
     "SPECTRUM ZABIERA  ";r
 430 PRINT AT 16,23;" "
 440 GO SUB 540
 450 PRINT AT 9,2;
     "ZOSTA£O ";p;" "
 460 IF p<1 THEN GO TO 480
 470 GO TO 240
 480 PRINT FLASH 1;AT 20,2;
     "    WYGRA£ KOMPUTER !!!
 490 BEEP 1,20: BEEP 2,8
 500 PAUSE 100: GO TO 10
 510 PRINT FLASH 1;AT 20,2;
     "       TY WYGRA£ES !!!
 520 BEEP .1,20: BEEP 2,20
 530 PAUSE 200: GO TO 10
 540 FOR i=1 TO r
 550 PRINT AT 12,p;"XXX"( TO r)
 560 BEEP .1,10*r
 570 PRINT AT 12,p;"   "( TO r)
 580 PAUSE 10
 590 NEXT i
 600 RETURN
 610 SAVE "ZABIERANKA" LINE 10
```

PROGRAM SHARP 29

Nasz program nazwaliśmy *„Czytanie myśli"*, gdyż może on sprawiać takie wrażenie.

Aby posługiwać się „odczytywaczem myśli", wystarczy znajomość... alfabetu: ściślej znajomość KOLEJNOŚCI LITER ALFABETU ŁACIŃSKIEGO.

Przykładowo na pytanie:
PRZED C ? T=1 N=0
należy odpowiedzieć „TAK", czyli wypaicować 1, gdy pomyślaną literą jest A lub B.

Jeżeli zaś pomyślaną literą jest C,D,E,...,Z, to należy odpowiedzieć „NIE", czyli 0. Uwaga: litera C NIE JEST PRZED C.

Po załadowaniu programu pomyślmy jakiś wyraz. Na początek krótki: UL, LAS, KOT.

Program wywołuje RUN „X". Od tego momentu bardzo uważnie odpowiadajmy na pytania. (Nasz jeden błąd psuje zabawę).

Gdy na pytanie
OSTATNIA T=1 N=0
czyli „czy to była OSTATNIA litera pomyślanego wyrazu" — gdy na pytanie odpowiemy „TAK" (1), wówczas komputer „odczyta pomyślany wyraz".

CZYTANIE... MYŚLI. Wykorzystuje się tu zasadę DYCHOTOMII (podziału na dwie przeciwstawne części). Najpierw całość alfabetu dzielimy na pół. Wskazaną przez odpowiedź połówkę dzielimy na ćwiartki. Potem na „ósemki, szesnastki i trzydziestkidwójki".

Ponieważ 2 × 2 × 2 × 2 × 2 = 32, zaś liter jest 26, zatem większość liter „odczytujemy" po PIĘCIU odpowiedziach, niektóre zaś po CZTERECH.

Życzymy dobrej zabawy!

JANUSZ MILLER
SHARP PC-1500

CZYTANIE MYSLI

```
600:"C":CLS
610:WAIT 0:PRINT "
    PRZED ";Z$;"?
    T=1 N=0 "
620:CURSOR 16:
    INPUT " ";D:
    CLS :RETURN
630:"D":USING
640:Z$="N":GOSUB "
    C":IF D=0GOTO
    820
650:Z$="H":GOSUB "
    C":IF D=0GOTO
    750
660:Z$="E":GOSUB "
    C":IF D=0GOTO
    720
670:Z$="C":GOSUB "
    C":IF D=0GOTO
    700
680:Z$="B":GOSUB "
    C":IF D=0LET X
    $="B":RETURN
690:X$="A":RETURN
700:Z$="D":GOSUB "
    C":IF D=1LET X
    $="C":RETURN
710:X$="D":RETURN
720:Z$="G":GOSUB "
    C":IF D=0LET X
    $="G":RETURN
730:Z$="F":GOSUB "
    C":IF D=0LET X
    $="F":RETURN
740:X$="E":RETURN
750:Z$="K":GOSUB "
    C":IF D=0GOTO
    790
760:Z$="J":GOSUB "
    C":IF D=0LET X
    $="J":RETURN
770:Z$="I":GOSUB "
    C":IF D=0LET X
    $="I":RETURN
780:X$="H":RETURN
790:Z$="M":GOSUB "
    C":IF D=0LET X
    $="M":RETURN
800:Z$="L":GOSUB "
    C":IF D=0LET X
    $="L":RETURN
810:X$="K":RETURN
820:Z$="T":GOSUB "
    C":IF D=0GOTO
    900
830:Z$="Q":GOSUB "
    C":IF D=0GOTO
    870
840:Z$="P":GOSUB "
    C":IF D=0LET X
    $="P":RETURN
850:Z$="O":GOSUB "
    C":IF D=0LET X
    $="O":RETURN
860:X$="N":RETURN
870:Z$="S":GOSUB "
    C":IF D=0LET X
    $="S":RETURN
880:Z$="R":GOSUB "
    C":IF D=0LET X
    $="R":RETURN
890:X$="Q":RETURN
900:Z$="W":GOSUB "
    C":IF D=0GOTO
    940
910:Z$="V":GOSUB "
    C":IF D=0LET X
    $="V":RETURN
920:Z$="U":GOSUB "
    C":IF D=0LET X
    $="U":RETURN
930:X$="T":RETURN
940:Z$="Y":GOSUB "
    C":IF D=0GOTO
    970
950:Z$="X":GOSUB "
    C":IF D=0LET X
    $="X":RETURN
960:X$="W":RETURN
970:Z$="Z":GOSUB "
    C":IF D=0LET X
    $="Z":RETURN
980:X$="Y":RETURN
 1000:"X":CLEAR :
      WAIT :CLS
 1010:K=K+1:WAIT :
      PRINT "LITER
      A";K
 1020:GOSUB "D":Y$
      =Y$+X$
 1030:INPUT "OSTAT
      NIA? T=1 N=0
       ";D:IF D=0
      GOTO 1010
 1040:WAIT :PRINT
      "KRYPTONIM "
      ;Y$:END
```

Zmienne systemowe ROM SPECTRUM

Dla tych, którzy poznali już BASIC, proponujemy spojrzenie w te sfery pamięci, które pozwolą wszechstronniej wykorzystać możliwości SPECTRUM. Dobra znajomość zmiennych systemowych jest wygodna dla programisty; ułatwia pracę i pozwala głębiej poznać system. Większość *sztuk i sztuczek*, jakie można wydobyć z tego obszaru, postaramy się przedstawić w kilku najbliższych numerach.

Klawiatura

KSTATE 23552..9
LAST— K 23560
REPDEL 23561
REPPER 23562
PIP 23009

Co 20 ms procedura przerwań maskowalnych o adresie 56 przeszukuje klawiaturę z tym samym celem — odszukania przyciśniętego klawisza. W przypadku pozytywnym dodatkowo następuje dekodowanie na system ASCII. Aktualny stan klawiatury zostaje umieszczony w grupie bajtów KSTATE, którą można podzielić na dwie części po 4 bajty. Oznaczmy adresy 23552...5, jako *B1, B2, B3, B4* zaś adresy 23556..9, jako *A1, A2, A3, A4*.

Wciskamy klawisz. Reaguje grupa druga *(A1...A4)*. Bajt *A1*, który miał dotąd wartość 255, co oznacza, że żaden klawisz nie był wciśnięty, szybko reaguje kodem naszego guzika. Przy czym jest to kod cyfry albo dużej litery. Nie ma więc znaczenia dla *A1*, czy przyciśnięto coś z CAPS SHIFT, czy SYMBOL SHIFT. Wyjątkiem jest reakcja na oba shifty (kod 14). Z chwilą przyciśnięcia pary innych klawiszy (oprócz obu shiftów) ładowany jest kod pierwszego wykrytego.

Natomiast A4 pokazuje już bardziej sprecyzowany kod ASCII. Uwzględnia, czy naciśnięto guzik z CAPS, czy SYMBOL SHIFT. Teraz wystarczy już tylko skupić się nad długością trwania nacisku. Z chwilą wykrycia klawisza w *A3* ładuje się wartość zmiennej systemowej REPDEL. Jest to czas mierzony w 50. częściach sekundy, po tym czasie klawisz zostanie uznany za przyciśnięty powtórnie. Standardowo ma ona wartość 35, czyli ok. 0,5 s. Z każdym przerwaniem *A3* jest dekrementowane (zmniejszane o 1), aż do zera. Komputer wie wtedy, że dalej trzymamy palec na klawiszu. Jeśli jesteśmy w edytorze, nastąpi drugi wydruk znaku o kodzie *A4* dla trybu małych liter (kursor L) lub o kodzie *A1* dla trybu dużych liter (kursor C). O tym, że komputer „czuje" nacisk palca, niech świadczy poniższy test:

```
10 POKE 23562,1
20 IF NOT LEN INKEY$ THEN GOTO 20
30 IF LEN INKEY$ THEN GOTO 30
40 IF PEEK 23558 +3 >= PEEK 23561 T
HEN PRINT "krotko": STOP
50 IF PEEK 23558 +11 >= PEEK 23561
THEN PRINT "srednio": STOP
60 PRINT "dlugo"
```

Kontynuując, dalsze samopowtarzanie zależy już od zawartości zmiennej systemowej REPPER. Decyduje ona o odstępie czasowym, w jakim ma powtarzać się uznawanie trzymanego klawisza za „ponownie wciśnięty" po wyzerowaniu zawartości REPDEL w A3. Normalnie jest tam wartość 5, co oznacza 0.1 s. Samopowtarzanie odbywa się na tej samej zasadzie co wcześniej, czyli na dekrementacji. Jeśli teraz A3 wyzeruje się, zawartość REPPER ładowana jest tam znowu itd. Gdy podczas tego procesu przyciśniemy inny klawisz, system zignoruje go, chyba że puścimy ten pierwszy. Wtedy nastąpi przerzut na komórki B1... B4 i cały omówiony proces powtarza się. W A1 pojawi się 255, zaś precyzyjny kod ostatnio przyciskanego klawisza zostaje umieszczony w zmiennej LAST—K i nie zmieniany, póki nie zostanie wykryty nowy klawisz.

Upraszczając, pierwsze powtórzenie guzika nastąpi po czasie zawartym w REPDEL, a każde następne po czasie zawartym w REPPER.

Należy podkreślić, że guzik naciśnięty z CAPS SHIFT lub SYMBOL SHIFT jest traktowany jako pojedynczy. Pozostaje do wyjaśnienia zawartość A2 lub B2. Jeśli nie naruszamy klawiatury, są tam zera, lecz po wykryciu nacisku pojawia się tam wartość 5 dekrementowana za każdym następnym przerwaniem. W praktyce oznacza to, że komputer 5 razy musi mieć potwierdzenie nacisku (0,1 s), i dopiero wówczas zareaguje dekodowaniem klawisza. Służbista? Nie tylko. Otóż zwarcie styków klawiatury powoduje ich nieznaczne drgania, co mogłoby spowodować błędy w wykryciu klawisza. Spokojne odczekanie przez system daje większą pewność użytkownikowi.

Poza otrzymywaniem szeregu informacji o stanie klawiatury, w zasadzie zmienna KSTATE nie jest wykorzystywana do innych celów. Zmiana wartości któregokolwiek bajtu nic nie da, gdyż uaktualnianie danych następuje piorunująco szybko. Zmieniać możemy wartości REPPER i REPDEL, modyfikując w ten sposób parametry samopowtarzania. Dobre efekty dla zaprzyjaźnionego już dobrze z klawiaturą daje:
POKE 23562,1
POKE 23561,30
POKE 23609,40
Ten ostatni adres (23609), to zmienna systemowa PIP. Decyduje ona o długości dźwięku wydanego z głośnika przy nacisku na klawisz. Początkowo zawiera 0, co objawia się cichym trzaskiem (ang. click). Podwyższenie o 1 powoduje zwiększenie czasu trwania dźwięku o ok. 1/765 s.

Na zakończenie kilka sztuczek. Poniższy program przełączający wydruk tekstu z ekranu na drukarkę i odwrotnie jest pretekstem do rezygnacji z dwuznacznej funkcji INKEY$:

```
10 PRINT "Drukarka / Ekran"
15 PAUSE 1: PAUSE 0
20 LET a= PEEK 23556
30 IF a= CODE "D" THEN LET s=3
40 IF a= CODE "E" THEN LET s=2
50 PRINT #s;"tekst"
```

Dla programistów zniechęconych niedostatkiem instrukcji PAUSE 0 mamy dobrą radę. Sekwencja:
10 PAUSE 0
20 IF PEEK 23558 < 35 THEN GOTO 10
działa znacznie lepiej.

Brak reakcji pojedynczych klawiszy SHIFT również można ominąć. Poniższy program zakończy się, jeśli rzeczywiście przyciśniemy dowolny klawisz

```
10 PRINT #1;AT 1,8;"dowolny klawisz
"
20 LET a= IN 254
30 IF a=255 OR a=191 THEN GOTO 20
40 STOP
```

W najbliższym numerze trochę szczegółów o pamięci RAM.

Krzysztof MAMCARZ

MOŻNA INACZEJ

Chciałbym podzielić się z Czytelnikami „IKS-a" kilkoma uwagami dotyczącymi przedstawionych w numerze 1/86 programów „Wykresy" i „Mapa Polski".

Wartość dydaktyczna programu „Wykresy" jest znacznie obniżona przez obarczenie użytkownika koniecznością podania wartości maksymalnej i minimalnej funkcji w rozpatrywanym przedziale. Wartości te — niezbędne do prawidłowego wyrysowania układu współrzędnych i opisu osi — mogą i muszą być obliczone przez komputer, na podstawie przepisu funkcji i przedziału zmienności zmiennej niezależnej x. Obliczenie tych wartości dla wielu niebanalnych funkcji nie jest rzeczą prostą, dlatego też zadanie to powinna wykonać maszyna. Podanie błędnego zakresu zmienności wartości funkcji powoduje całkowite zniekształcenie wykresu (dla przykładu proszę sprawdzić: y = SIN X, XMIN = 0, XMAX = 5, YMIN = 0, YMAX = 2).

Przedstawiony program nie sporządzi wykresu funkcji stałej y(x) = a. Funkcja stała, aczkolwiek bardzo prosta, musi być przez program akceptowana. Program nie poradzi sobie również z funkcjami nieciągłymi w zadanym przedziale, np. y(x) = 1/x czy y(x) = (SIN X).

W przypadku podania przedziału zmienności argumentu x mniejszego od 0.5 np. (0.1, 0.2) wygenerowany opis osi będzie błędny. Prawidłowe działanie programu dla małych długości przedziału jest niezbędne, ponieważ umożliwia oglądanie wycinków wykresu funkcji.

W programie nie są sprawdzane dane wejściowe XMIN, XMAX, YMIN, YMAX (powinny to być liczby). Bardzo łatwo można więc doprowadzić do wystąpienia błędu, podając — omyłkowo lub świadomie — wielkości nie będące liczbami.

Sprawdzenie warunku w instrukcji 460

IF CODE Z$ = 84 OR CODE Z$ = 116...

można uprościć pisząc:

IF Z$ = „T" OR Z$ = „t"...

Ponadto należałoby najpierw sprawdzić, czy wprowadzony znak jest jednym ze znaków T, t, N, n. Milczące założenie, że wszystkie znaki różne od T i t są równoznaczne z N/n jest błędem, gdyż prawdopodobieństwo błędnego napisania znaku N lub n jest równe prawdopodobieństwu błędnego wpisania znaku T lub t.

W programie „Mapa Polski" do wykreślenia konturów mapy wielokrotnie użyto instrukcji DRAW z trzema parametrami (łuki), co znacznie wydłuża czas rysowania choć program czyni czytelnym. Poniżej przedstawiam swoją wersję rysowania mapy, która — choć trochę dłuższa — jest kilkadziesiąt razy szybsza (czas rysowania mniejszy od 0.5 s).

Ponadto w obydwu programach po zakończeniu obliczeń nie przywrócono „warunków normalnych", tj. BORDER 7 : PAPER 7 : INK 0 : CLS. Pozostawienie po wyjściu z programu takich warunków powinno być regułą, aby nie powodować ewentualnych zakłóceń w pracy programu następnego użytkownika.

Janusz MORBITZER

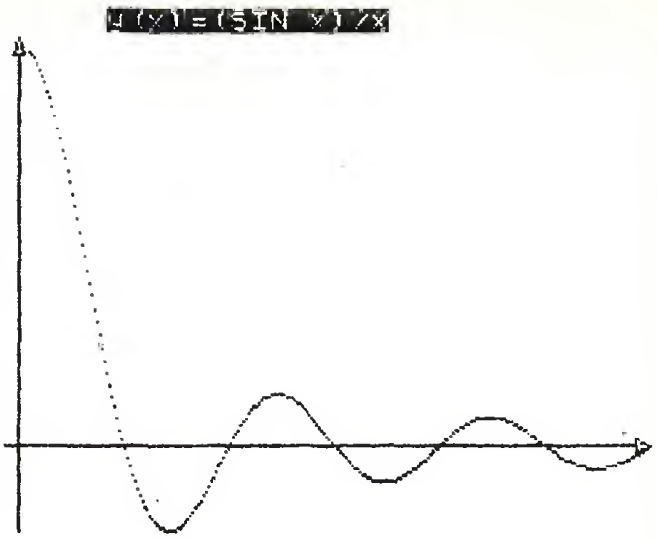
```
5000 REM rysowanie mapy Polski
5010 CLS : PLOT 138,160: DRAW 27
,-3: DRAW 3,3: DRAW 9,-5: DRAW 8
,-26: DRAW 1,-16: DRAW -4,-2: DR
AW -6,-8: DRAW 8,-5
5020 DRAW -2,-8: DRAW 3,-10: DRA
W 3,-2: DRAW 0,-2: DRAW 5,-5: DR
AW -3,-2: DRAW 3,-4: DRAW -1,-5:
 DRAW -6,-1: DRAW -15,-17: DRAW
2,-10: DRAW 3,-2: DRAW -2,-2
5030 DRAW -12,4: DRAW -1,2: DRAW
 -12,1: DRAW -3,-3: DRAW -3,2: D
RAW -5,0: DRAW -3,-2: DRAW -1,-4
: DRAW -2,3: DRAW -2,-2: DRAW 0,
5: DRAW -4,0: DRAW -2,3
5040 DRAW -3,-2: DRAW -1,-3: DRA
W -6,0: DRAW -8,11: DRAW -8,4: D
RAW -4,-2: DRAW -4,5: DRAW 3,2:
DRAW -1,2: DRAW -8,1
5050 DRAW -4,4: DRAW -2,-1: DRAW
 3,-7: DRAW -6,-3: DRAW -6,10: D
RAW 4,3: DRAW -1,2: DRAW -8,-1:
DRAW -1,2: DRAW -3,0: DRAW -2,2:
 DRAW -3,-1: DRAW -1,7
5060 DRAW -3,0: DRAW -1,-4: DRAW
 -6,1: DRAW 2,3: DRAW 2,11: DRAW
 -5,3: DRAW 1,1: DRAW -3,4: DRAW
 3,4: DRAW 0,8: DRAW -3,1: DRAW
0,6: DRAW 1,0: DRAW -2,8
5070 DRAW -7,6: DRAW 5,6: DRAW -
2,20: DRAW 28,9: DRAW 8,5: DRAW
6,1: DRAW 6,4: DRAW 16,3: DRAW 8
,-1
5080 DRAW 8,-6: DRAW -2,-2: DRAW
 -4,4: DRAW 1,-6: DRAW 5,-3: DRA
W 5,1: DRAW 5,3: DRAW 12,-1
5090 RETURN
```

Program „Wykresy"

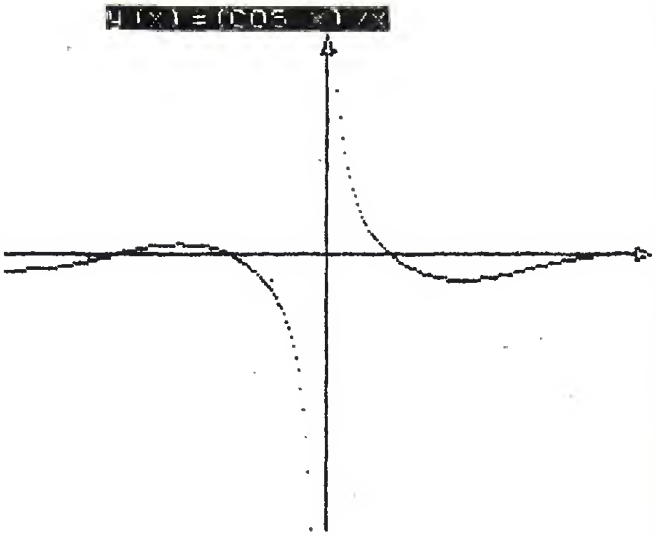
na stronie 10

```
    1 REM program "Funkcja"
    2 REM Autor-Janusz MORBITZER
    5 BEEP .2,15: PAUSE 30: BORDE
R 7: PAPER 7: INK 0: CLS
    7 POKE 23609,50
   15 CLS : PRINT BRIGHT 1;AT 8,8
;"PROGRAM ""Funkcja""": PRINT BR
IGHT 1;AT 13,5;"Autor: Janusz MO
RBITZER": PRINT AT 16,11; BRIGHT
 1;"WSP-KRAKÓW"
   20 DIM x(250): DIM f(250): DIM
 p(2)
   22 GO SUB 600: CLS : PRINT AT
8,3;"Program ""Funkcja"" realizu
je": PRINT AT 9,6;"wydruk funkcj
i y=f(x)": PRINT AT 10,8;"w prze
dziale <a,b>": PRINT AT 11,6;"or
az oblicza maksimum ": PRINT AT
12,8;"i minimum funkcji": PRINT
AT 13,8;"w tym przedziale": GO S
UB 550
   25 LET p(1)=1000000: LET p(2)=
1000000
   27 CLS : PRINT AT 14,5; BRIGHT
 1;"Podaj przepis funkcji:"
   30 INPUT "f(x)= "; LINE k$: IF
 LEN k$=0 THEN GO TO 30
   35 LET dlugosc=5+LEN k$: LET p
ozy=(31-dlugosc)/2
   40 CLS : PRINT BRIGHT 1;AT 14,
8;"Podaj przedział:"
   41 INPUT "  a="; LINE a$;"
       b="; LINE b$: IF LEN a$=0
OR LEN b$=0 THEN GO TO 41
   43 FOR i=1 TO LEN a$: IF a$(i)
>="0" AND a$(i)<="9" OR a$(i)=".
" OR a$(i)="-" OR a$(i)="+" OR a
$(i)="PI" OR a$(i)="*" OR a$(i)=
"/" THEN NEXT i: GO TO 50
   44 GO TO 40
   51 FOR i=1 TO LEN b$: IF b$(i)
>="0" AND b$(i)<="9" OR b$(i)=".
" OR b$(i)="-" OR b$(i)="+" OR b
$(i)="PI" OR b$(i)="*" OR b$(i)=
"/" THEN NEXT i: GO TO 55
   52 GO TO 40
   55 LET a=VAL a$: LET b=VAL b$
   60 IF a>=b THEN GO TO 40
   61 LET przedział=0
   62 LET dt=b-a: LET r=SGN (a*b)
   64 IF r=1 AND a>0 THEN LET dt=
b: GO TO 68
   66 IF r=1 AND a<0 THEN LET dt=
ABS (a)
   68 IF dt<>b-a THEN LET przedzi
ał=1
   70 LET z=ABS (a): LET d=dt/250
: LET d4=4*d
   80 CLS : PRINT BRIGHT 1;AT 14,
6;"Podaj liczbę punktów": PRINT
AT 16,6; BRIGHT 1;"nieciągłości
 (0-2):"
   82 LET w$=INKEY$: IF LEN w$<>1
 THEN GO TO 82
   90 IF w$<"0" OR w$>"2" THEN GO
 TO 82
  100 LET w=VAL w$: IF w=0 THEN G
O TO 140
  105 FOR i=1 TO w
  110 PRINT AT 21,3;"Podaj ";i;".
pkt nieciągłości:"
  115 INPUT LINE c$: IF LEN c$=0
THEN GO TO 115
  120 FOR m=1 TO LEN c$: IF c$(m)
>="0" AND c$(m)<="9" OR c$(m)=".
" OR c$(m)="-" OR c$(m)="*" OR c
$(m)="+" OR c$(m)="PI" THEN NEXT
 m: LET p(i)=VAL c$: GO TO 130
  125 GO TO 115
  130 IF NOT (p(i)>=a AND p(i)<=b
) THEN LET w=w-1
  135 NEXT i
  137 LET pn1=p(1): LET pn2=p(2)
  140 LET n=249: LET l=0
  145 BORDER 0: PAPER 0: INK 7: L
ET dk=(28-LEN a$-LEN b$)/2: CLS
: PRINT AT 4,9; FLASH 1;"PROSZE
 CZEKAĆ": PRINT AT 8,8;"Obecnie f
unkcja:": PRINT INVERSE 1;AT 10,
pozy;"y(x)=";k$: PRINT AT 12,0;"
jest tabelaryzowana w przedziale
": PRINT INVERSE 1;AT 14,dk;"[";
a$;",";b$;"]": PRINT AT 16,0;"Kr
ok tabelaryzacji dx=";d
  150 FOR i=1 TO n+1
  160 LET x=a+(i-1)*d: IF x>b THE
N GO TO 210
  170 IF w=0 THEN GO TO 190
  180 IF ABS (x-pn1)<d4 OR ABS (x
-pn2)<d4 THEN GO TO 200
  190 LET l=l+1: LET x(l)=x: LET
f(l)=VAL k$
  200 NEXT i
  210 LET fmin=f(1): LET fmax=f(1
)
  220 LET k1=1: LET k2=1
  230 FOR i=2 TO l
  240 IF (f(i)-fmin)>=0 THEN GO T
O 260
  250 LET fmin=f(i): LET k1=i: GO
 TO 280
  260 IF (f(i)-fmax)<=0 THEN GO T
O 280
  270 LET fmax=f(i): LET k2=i
  280 NEXT i
  290 LET xmin=x(k1): LET xmax=x(
k2)
  295 LET m$=STR$ fmax: IF LEN m$
>9 THEN LET m$=m$( TO 9)
  296 LET n$=STR$ fmin: IF LEN n$
>9 THEN LET n$=n$( TO 9)
  297 LET x$=STR$ xmin: IF LEN x$
>9 THEN LET x$=x$( TO 9)
  298 LET y$=STR$ xmax: IF LEN y$
>9 THEN LET y$=y$( TO 9)
  300 LET dy=fmax-fmin: BEEP .8,1
5: BORDER 7: PAPER 7: INK 0: CLS

  310 PRINT BRIGHT 1;AT 9,0;"FMIN
=";n$;" dla x=";x$
  320 PRINT BRIGHT 1;AT 11,0;"FMA
X=";m$;" dla x=";y$
  330 GO SUB 550
  340 CLS : LET s=SGN (fmin*fmax)
  350 IF s=1 AND fmin>0 THEN LET
y=5: GO TO 400
  360 IF s=1 AND fmin<0 THEN LET
y=160: GO TO 400
  370 IF s=0 AND fmax>0 THEN LET
y=5: GO TO 400
  380 IF s=0 AND fmax=0 THEN LET
y=160: GO TO 400
  390 LET y=INT ((ABS (fmin)/dy)*
165): IF y<5 THEN LET y=5
  395 IF y>160 THEN LET y=160
  400 LET t=SGN (a*b)
  410 IF t>=0 AND a>=0 THEN LET x
=5: GO TO 440
  420 IF t>=0 AND a<0 THEN LET x=
249: GO TO 440
  430 LET x=INT ((z/dt)*255)
  440 IF x<250 THEN PLOT 0,y: DRA
W 249,0: PLOT 249,y+2: DRAW 0,-4
: DRAW 6,2: DRAW -6,2: GO TO 460
  450 PLOT 6,y: DRAW 249,0: PLOT
6,y+2: DRAW 0,-4: DRAW -6,2: DRA
W 6,2
  460 IF y<160 THEN PLOT x,0: DRA
W 0,159: PLOT x-2,159: DRAW 4,0:
 DRAW -2,6: DRAW -2,-6: GO TO 47
8
  470 PLOT x,0: DRAW 0,159: PLOT
x-2,6: DRAW 4,0: DRAW -2,-6: DRA
W -2,6
  478 IF dy<>0 THEN LET z3=160/dy
  480 LET poprx=0: LET popry=0: L
ET z1=250/dt: IF fmin<>0 THEN LE
T z4=160/fmin
  481 IF x=5 THEN LET poprx=5
  482 IF y=5 THEN LET popry=5
  483 IF fmax=0 THEN GO TO 485
  484 LET z2=160/fmax
```

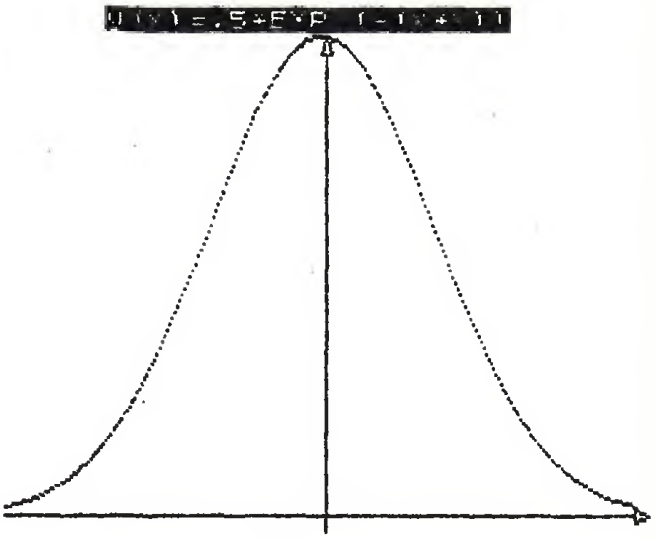
```
 485 IF przedzial=1 AND a>0 THEN
 LET z=0
 487 IF s>=0 AND fmax>0 THEN LET
 f$="f(i)*z2+poprY": GO TO 490
 488 IF s>=0 AND fmin<0 THEN LET
 f$="160-ABS (f(i)*z4)": GO TO 4
90
 489 LET f$="ABS ((f(i)-fmin)*z3
)+poprY"
 490 PRINT AT 0,5; INVERSE 1;"y(
x)=";k$
 495 FOR i=1 TO l
 500 LET x=((z+x(i))*z1+poprX)
 510 LET y=VAL f$
 515 IF ABS x>255 OR ABS y>175 T
HEN GO TO 530
 520 INK 0: PLOT x,y
 530 NEXT i
 531 PRINT #1;AT 0,2; BRIGHT 1;"
Zadany wydruk(t-tak,n-nie):": LE
T w$=INKEY$: IF LEN w$=0 THEN GO
 TO 531
 532 IF w$<>"t" AND w$<>"n" THEN
 GO TO 531
 533 IF w$="t" THEN COPY : LPRIN
T : LPRINT "Przedział: [";a$;",
";b$;"]": LPRINT "Fmin=";n$;" dla
 x=";x$: LPRINT "Fmax=";m$;" dla
 x=";y$
 535 PRINT #1;AT 0,0;"
                      ": PRINT #
1;AT 0,1; BRIGHT 1;"Podaj(0-koni
ec,1-kontynuacja):"
 536 LET t$=INKEY$: IF LEN t$=0
THEN GO TO 536
 538 IF t$<>"0" AND t$<>"1" THEN
 GO TO 536
 539 IF t$="1" THEN GO TO 25
 540 CLS : PRINT AT 8,10;"******
******": PRINT AT 9,10;"*
   *": PRINT AT 10,10;"*  KONIEC
  *": PRINT AT 11,10;"* PROGRAMU
 *": PRINT AT 12,10;"*
*": PRINT AT 13,10;"************
": GO SUB 600: GO TO 630
 545 REM podprogram
 550  PRINT #1; BRIGHT 1; FLASH
1;AT 0,4;"Naciśnij dowolny klawi
sz"
 560 PAUSE 0: RETURN
 600 REM podprogram
 610 LET t=.25: BEEP t,0: BEEP t
,2: BEEP t,4: BEEP t,5: BEEP t,7
: BEEP t,9: BEEP t,11: BEEP t,12
: PAUSE 10: BEEP t,12: BEEP t,11
: BEEP t,9: BEEP t,7: BEEP t,5:
BEEP t,4: BEEP t,2: BEEP t,0
 620 RETURN
 630 PAUSE 0: BORDER 7: PAPER 7:
 INK 0: CLS : GO TO 15: REM koni
ec programu
```




```
Przedział: [0,6*PI]
Fmin=-0.217133 dla x=4.5238934
Fmax=0.9849090 dla x=0.3015928
```




```
Przedział: [-2*PI,2*PI]
Fmin=-4.873399 dla x=-0.201061
Fmax=3.8538699 dla x=0.2513274
```




```
Przedział: [-2,2]
Fmin=.00915781 dla x=-2
Fmax=0.5 dla x=0
```

UWAGA !

„IKS" w prenumeracie! Prenumeratę przyjmują oddziały RSW „Prasa-Książka-Ruch", a w miejscowościach, gdzie ich nie ma urzędy pocztowe i doręczyciele.

METODA TRAPEZÓW

Znalezienie wartości całki oznaczonej zadanej funkcji w określonym przedziale na ogół nie jest trudne. Pierwszym, najtrudniejszym krokiem jest znalezienie funkcji pierwotnej — czynności następne są już działaniami arytmetycznymi, nie sprawiają większych kłopotów. Prawdziwe zmartwienie zaczyna się wtedy, kiedy funkcja podcałkowa nie posiada funkcji pierwotnej. Takl problem mamy z funkcją $\exp(-x^2)$, której całkę oznaczoną możemy obliczyć jedynie metodami numerycznymi

Możemy to zrobić metodą Newtona-Cotesa, Simpsona albo metodą trapezów. Wybierzmy tę ostatnią. Przedział całkowania dzielimy na n równych części, w konsekwencji otrzymujemy węzły:

$x_0 = a < x_1 < x_2 < x_3 < \ldots < x_{n-1} < x_n = b$

Jako przybliżenie całki w metodzie trapezów przyjmuje się wartość następującego wyrażenia:

$S = h(^1/_2 y_0 + y_1 + y_2 + \ldots y_{n-1} + ^1/_2 y_n)$

gdzie:

$y_i = f(x_i)$ $(i = 0, 1, 2, \ldots n)$

f — funkcja podcałkowa

h — wysokość elementarnego trapezu (jednakowa dla wszystkich trapezów, co wynika z podziału przedziału całkowania na n równych części).

Metoda ta nie jest wolna od błędu, jest on określony wzorem:

$$R = \frac{-(b-a)h^2 y'' t}{12}, \text{ dla } t \in \langle a, b \rangle$$

Przystępując do obliczania całki metodą trapezów nie wiemy na ile części podzielić całkowania $\langle a, b \rangle$, tak, aby wynik mieścił się w zadanej dokładności eps. Musimy to uwzględnić w realizowanym algorytmie.

Krok pierwszy. Jako pierwsze przybliżenie (początkowe) całki przyjmujemy wartość całki obliczoną dla n = 1. Jest ono równe polu trapezu o bokach f(a), f(b) i wysokości h = b — a.

Krok drugi. Następne przybliżenie otrzymujemy przez podwojenie liczby podziałów przedziału $\langle a, b \rangle$ z poprzedniego przybliżenia. Przedział całkowania dzielimy oczywiście na równe części. Liczba podziałów jest podwajana, a wysokość h zmniejsza się o połowę.

Krok trzeci. Z otrzymanych podziałów obliczamy kolejne przybliżenie całki. Sprawdzamy, czy moduł różnicy pomiędzy tym i poprzednim przybliżeniem jest mniejszy od zadanej wielkości eps. Jeżeli nie, to cofamy się do kroku drugiego, jeżeli tak, to w tym miejscu następuje koniec obliczeń i wydruk wyników.

```
comment Obliczanie całki metodą trapezów
begin
    real h, a, b, eps, x; x0, x1, y, s;
    integer i, n;
    line (1);
    print ('WARTOŚĆ CAŁKI W
            PRZEDZIALE < a, b >');
    line (1);
P: read (a, b, eps);
    h = b — a; n = 1;
    s = (exp(—a↑2) + exp(—b↑2))/2;
    x0 = s * h;
    y = 10 ↑ 2;
    for n = 2 * n while y > eps do
    begin
h = h/2; x1 = s;
x = a;
for i = 1 step 1 until n — 1 do
begin
    x = x + h;
    x1 = x1 + exp(—x↑2)
end i;
x1 = x1 * h;
y = abs(x1 — x0);
x0 = x1
    end n;
    format (' —12.346  '); line (2);
    print (' a = ', a, ' b = ', b, ' eps = ',
            eps, ' wartość całki = ', x1);
    wait (' nowe dane ');
    goto P
end programu
```

Algorytm zakodowany został w dwóch językach: Algolu i w dobrze nam znanym Basicu. Łatwo wyróżnić w nich sekwencje wykonujące te same obliczenia. Algol jest językiem, w którym zapis programu pozostaje w dużej zgodności z „kształtem" algorytmu — bardzo często zamiast rysować schemat działania rozwiązanie problemu zapisuje się wprost w Algolu.

Różnice? Program w tym języku wymaga wcześniejszych deklaracji zmiennych i stałych. **Real** to liczby rzeczywiste **integer** — całkowite. Komenda line (1) powoduje wysunięcie papieru drukarki o jeden wiersz. Inne są oczywiście komendy i formaty instrukcji we/wy.

Właściwości Algolu pozwoliły na warunkowe zorganizowanie pętli:

for n = 2 * n **while** y > eps **do**

instrukcje z jej wnętrza wykonywane są w przypadku gdy y > eps. Aby zapewnić sobie przynajmniej jednokrotne ich wykonanie przed pętlą dokonano podstawienia y = 100 — eps z pewnością jest mniejsze.

W Basicu natomiast warunek ten sprawdzany jest w wierszu — 200 — w ten sposób, bez uprzednich podstawień, obliczenia pierwszego przybliżenia wartości całki będą zawsze wykonane.

```
 10 REM Obliczanie całki metodą trapezów
 15 PRINT „WARTOŚĆ CAŁKI W PRZEDZIALE < a, b >"
 20 INPUT „podaj dolną granicę całkowania"; a
 30 INPUT „podaj górną granicę całkowania"; b
 35 INPUT „podaj dokładność": eps
 40 LET h = b — a
 50 LET n = 1
 60 LET s = (EXP(—a↑2) + EXP(—b↑2))/2
 70 LET x0 = s * h
 90 LET n = 2 * n
100 LET h = h/2
110 LET x1 = s
120 LET x = a
125 REM Początek pętli warunkowej
130 FOR i = 1 TO n — 1
140 LET x = x + h
150 LET x1 = x1 + EXP(—x↑2)
160 NEXT 1
170 LET = x1 = x1 * h
180 LET y = ABS(x1 — x0)
190 LET x0 = x1
200 IF y > = eps THEN GOTO 90
210 PRINT "a = "; a, "b = "; b, "eps = "; eps, "wartość całki = "; x1
220 LET b$ = "TAK"
230 INPUT "czy nowe dane TAK/ /NIE"; a$
240 IF b$ = a$ THEN GOTO 20
250 STOP
```

A.S.

Jedno z komputerowych pism angielskich nie bez złośliwości stwierdza, że trudno jest podziwiać francuski mikrokomputer bez świadomości, że wytwarzane we Francji produkty są niepowtarzalne: sery, wina, samochody i... komputery.

Tylko Francuzi mogli stworzyć wehikuł (CITROEN 2CV) o uroku, który niesie sobą brak elegancji. Tylko oni mogli rozwinąć mikro typu GOUPIL G4. G4 pracuje pod systemem MS DOS i praktycznie jest kompatybilny z IBM. Oceniając ten mikro, należy mieć pewien pogląd o genealogii maszyny i firmie, która ten komputer zbudowała. SOCIETE DE MICRO — INFORMATIQUE lub prościej SMT International. Firma ta ma zaledwie pięć lat, ale już od ponad dwóch lat stara się zdobyć rynek angielski. Oczywiście, towarem jest mikro typu GOUPIL. Słowo to według średniowiecznej francuskiej legendy oznacza sympatycznego drapieżnika — LISA.

Do końca 1983 roku SMT sprzedał nieco poniżej 10 000 tych maszyn. Potem firma zdecydowała się na wspomnianą próbę zdobycia rynku angielskiego. Proponowano tam komputery w trzech wersjach różniących się procesorami: 6809, Z-80, 8088. Oferowane maszyny umożliwiały między innymi wieloprogramowość.

Niestety, wynik był mizerny. Starano się więc wykorzystać bardziej konwencjonalny komputer G4. Wykorzystuje on standardową (nieco zmodyfikowaną) klawiaturę IBM. Klawisze funkcyjne zostały przesunięte z lewej strony nad klawisze alfanumeryczne.

Warto w tym miejscu podkreślić olbrzymie znaczenie usytuowania klawiszy, szczególnie funkcyjnych. Nawet pozornie drobne zmiany przyjętych standardów (praktycznie narzuca je IBM) mogą znacznie utrudnić wykorzystanie komputera. Tak jest w wypadku G4, bowiem część klawiszy jest źle usytuowana; trudno na przykład znaleźć klawisz BREAK.

W mikro G4 zainstalowano dyski typu IBM, a w skład jego wyposażenia wchodzi „myszka" i długo oczekiwany system Microsofta OKNA (WINDOWS). Serce G4 to 16-bitowy mikroprocesor 80186.

80166 jest zastanawiającym wyborem, jest różny bowiem od 8088 wykorzystanym w IBM PC, a jednocześnie ma liczne braki wobec 80286, wykorzystanego w IBM PC AT.

G4 realizuje oprogramowanie IBMa, Electric Disck, Chit-Chat i Framework w większości wypadków robi to szybciej niż konwencjonalne maszyny IBM.

Oceniany model ma pamięć 256 KB RAM, twardy dysk 10 MB i kompatybilne do IBM floppy. Typowe uzupełnienie to monitor oraz klawiatura. Jednak o atrakcyjności G4 decydują OKNA Microsofta i doskonale oprogramowana „myszka".

Daje się odczuć, że jakość konstrukcji komputera, jego oprogramowanie wiąże się z odpowiednim przygotowaniem rynku, zaspokojeniem najbardziej wyszukanych potrzeb potencjalnych klientów, łącznie z poradnictwem np. w zakresie instalacji komputera w wybranej firmie.

SMT zdaje sobie sprawę, że nie zdobędzie rynku jedynie niskimi cenami. Jest tu zbyt wielka konkurencja w postaci takich firm jak TANDY, COMMODORE. Goupil oferuje trochę większą szybkość, nieco lepszy styl oraz oczekiwane, bardzo praktyczne OKNA.

OKNA są realizacją i jednocześnie zwiastunem nowej, niewątpliwie trwałej tendencji rozwoju i doskonalenia *„interfejsu użytkownika"*. Tych możliwości komputera, które decydują o wzroście *„życzliwości komputera"*, prostszym i bardziej naturalnym sposobie komunikowania się z komputerem.

Komputer G4 jest przyjemny w eksploatacji i nie zaskakuje doświadczonych użytkowników PC, aczkolwiek podczas pracy zestawu pojawia się czasem błąd parzystości.

Jedynie monochromatyczny ekran nieco zaskakuje zbyt małą rozdzielczością, jakiej należałoby oczekiwać w maszynie, która ma się stać popularna właśnie dzięki grafice użytkowej.

Interfejsowe możliwości G4 wydają się standardowe. Oba szeregowe i równoległe porty są wbudowane w tylnej części. Pracują poprawnie z modemami i równoległą drukarką. System twardego dysku jest cichy i szybki — sprawia wrażenie, że to dodatkowa pamięć RAM.

Pamięć 640 KB powinna być wystarczająca dla większości zastosowań. Należy jednak sprawdzić, ile OKNA zostawiają pamięci dla programów użytkowych, przed określeniem ostatecznej konfiguracji G4 (cena 1395 funtów 128 KB lub 4420 funtów 512 KB).

Część przedstawionych powyżej uwag na temat jednego z ostatnich francuskich komputerów tylko pozornie jest nieco różna od informatycznych zainteresowań czytelników „IKS-a". Jutro bowiem istotną kwestią stanie się wybór komputera dla naszego zakładu pracy, wyróżnienie jego podstawowych cech, które decydują o jakości i komforcie pracy maszyny. Wydaje się, że zasadniczym wnioskiem jest uznanie kompatybilności z IBM jako podstawowej cechy profesjonalnego mikrokomputera. Kolejną cechą, którą wyróżniono jest „życzliwość komputera" (friendliness). Mieści się w tym zarówno mechaniczna konstrukcja, jak i oprogramowanie maszyny.

Na przykładzie G4, „życzliwość" swą mikro uzyskuje dzięki bardzo komunikatywnemu systemowi OKNA. Uzupełnieniem tej własności miała być nieudana próba poprawienia konstrukcji klawiatury (rozmieszczenia klawiszy).

Wymowne jest to, że profesjonalne pismo komputerowe bardziej eksponuje składowe „życzliwości komputera", na przykład kiepską jakość monitora, niż szybkość G4, która przekracza możliwości IBM.

Można sądzić, że wkraczamy w nowy etap rozwoju mikrokomputerów, etap dążeń ku poprawie jakości współpracy z komputerami w pracy i w domu.

W.G.

Giełda na Grzybowskiej... wrzesień'86

Foto: Jan Zelman

Jak to zrobić (1)

Zapewne wszystkich fascynuje barwny, ruszający się świat gier. Dociekliwi, zanim wykonają pierwszy ruch joystick'iem, zastanawiają się, JAK TO można ZROBIĆ. Artykuł ten wyjaśnia podstawy najprostszej techniki animacji na ATARI 800, jaką dostarcza grafika PLAYER-MISSILE, odpowiadająca SPRIT'om na innych komputerach.

W związku z tym, że w dalszej części używane będą operacje POKE i PEEK, a nie chcę osobom rozpoczynającym swoją przygodę z komputerem odbierać szansy zrozumienia, rozpocząć muszę od kilku uwag z nimi związanych.

PEEK jest funkcją, która odczytuje wartość bajtu spod wskazanego adresu. Na przykład instrukcje

```
A = PEEK (704)
B = PEEK (53248)
```

przypisują zmiennym A i B wartości zapamiętane odpowiednio w komórkach o adresach 704 i 53248.

Odwrotne role odgrywa procedura POKE. Pod adresem występującym w roli jednego z jej parametrów umieszcza ona dowolną wartość z przedziału 0—255, podana jako drugi parametr. Na przykład

```
POKE 704,160
POKE 53248,100
```

Przechodząc już do właściwego tematu, należy odpowiedzieć na pytanie, czym jest *player*?

Wyobraźmy sobie pionowy pasek na ekranie. Podobnie jak wszystko, co jest wyświetlane, pasek nasz jest obrazem pewnego kawałka pamięci komputera. Jeżeli teraz (... na razie) uwierzymy, że może się on przesuwać wzdłuż ekranu, zmieniać swoją szerokość i barwę, to będziemy znali najprostszą odpowiedź na to pytanie.

Należy dodać, że dysponujemy czterema *playerami* i czterema *misselami*, z których każdy jest także „paskiem", ale o szerokości 1/4 rozmiaru *playera*.

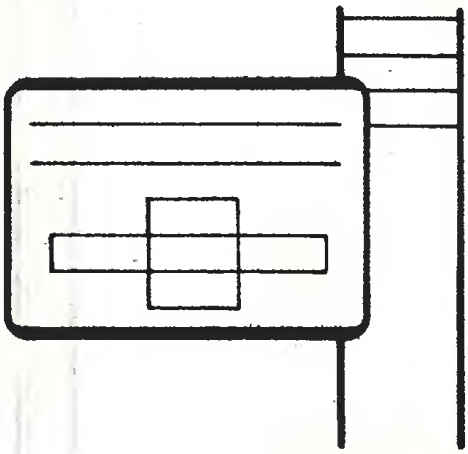
Wielu zapalonych amatorów programowania już wyobraża sobie bajeczne obrazy własnego autorstwa, przesuwające się po ekranie. My jednak postępować powinniśmy systematycznie, a w fazie poznawania możliwości *playerów* powoli, ale dokładnie analizować każdy krok.

Rozszyfrowywanie tajemnic grafiki PLAYER-MISSILE rozpoczniemy od następującego stwierdzenia:

player jest obrazem ciągłego obszaru pamięci.

Fakt ten ma ogromne znaczenie przy definiowaniu kształtu *playera*, co zobaczymy dalej. Rola, jaką on odgrywa, i cała „filozofia" *playerów* szczególnie widoczna jest w momencie analizy sposobu, w jaki komputer wyświetla zadane informacje na ekranie.

Wiemy, że obraz widoczny na monitorze jest odwzorowaniem pewnej części pamięci. W zależności od wybranej grafiki, różne jej komórki mogą być wyświetlane w różnych miejscach ekranu (zależy to od liczby bajtów tworzących linię dla danej grafiki). Różny jest także sposób liczenia adresów bajtów wyświetlanych bezpośrednio wyżej i niżej wybranej pozycji ekranu. Dla grafiki „0" adresy pamięci, tworzącej obraz ekranu, przedstawić można następująco:



Realia te powodują, że stworzenie pionowego obrazu, który mógłby się poruszać, jest, może nietrudne, ale dość skomplikowane i pracochłonne. Ponadto zauważmy, że każdorazowa chęć zmiany grafiki niesie ze sobą niebezpieczeństwo zniszczenia rysunku.

Rozwiązanie tego problemu dostarcza grafika *playerów*. Ciągłość obszaru pamięci przeznaczonej dla ich reprezentacji i jego niezależność od pamięci ekranu zapewniają:

— łatwość dostępu do kolejnych bajtów, odpowiadających następującym po sobie liniom obrazu,
— uniezależnienie *playera* od trybu graficznego tła.

Ponadto komputer dostarcza nam możliwość wyboru jednego z dwóch sposobów odwzorowania pamięci *playera* na ekran różniących się rezolucją, tj. liczbą linii opisanych jednym bajtem. Możemy zażądać, by każdy bajt przeniesiony został na dwie linie ekranu — rezolucja podwójna, ale wyraźniejszy rysunek uzyskujemy przy rezolucji pojedynczej — każdy bajt tworzy jedną linię obrazu.

Do wskazania wybranej rezolucji służy czwarty bit komórki o adresie 559. Brzmi to przerażająco, dlatego należy natychmiast zapomnieć to zdanie, a zapamiętać dwie liczby 62 i 46, które wpisane instrukcją

```
POKE 559,...
```

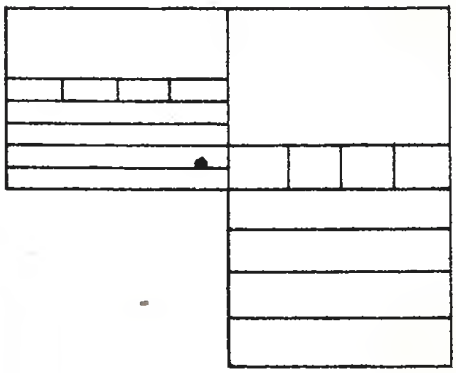
zapewniają

62 — rezolucja pojedyncza,

46 — rezolucja podwójna.

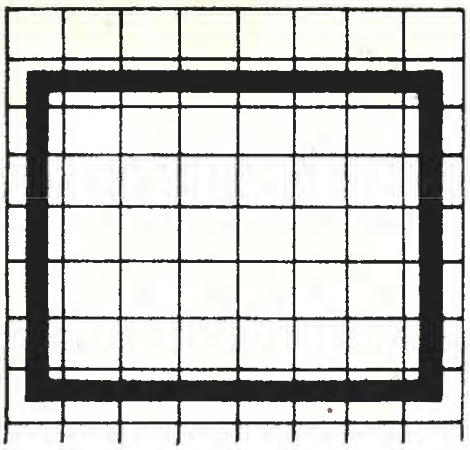
Z rezolucją *playerów* ściśle związana jest wielkość pamięci, potrzebna do ich reprezentacji, dla pojedyńczej wynosi aż 256 bajtów, dla podwójnej tylko połowę, czyli 128 bajtów. Dla poprawnej obsługi *playerów* adres tego pola znajdować się musi w komórce o adresie 54279. Może to być oczywiście liczba nie przekraczająca 255 (dlaczego?), dlatego właściwy adres wyraża ona w postaci bloków po 256 bajtów każdy.

Poniższy rysunek prezentuje schemat omawianej pamięci.



Po tych niezbędnych wyjaśnieniach przystąpić możemy do nadania *playerowi* kształtu. Metoda jest powszechnie znana — rysujemy planowany obraz na siatce o szerokości ośmiu pól, w której kolumny, począwszy od prawej strony, ponumerowane są kolejnymi potęgami dwójki 1,2,4,... Następnie pola, przez które przechodzą linie rysunku, zamieniamy na 1, pozostałe na 0. Dla każdej linii rysunku obliczamy wartość będącą sumą iloczynów wartości pola (1 lub 0) przez wartość kolumny, w której się znajduje (1,2,4,...).

Zasadę ilustruje poniższy przykład:



Szczególną ostrożność należy zachować przy tworzeniu rysunków *misseli*, bowiem jeden bajt pamięci zawiera informację o postaci pojedynczej linii każdego z nich. I tak na powyższym rysunku kolumny (bity) z numerami

1,2 — zawierają informacje dotyczące *missela* 0,

4,8 — dotyczą *missela* 1,

16,32 — *missela* 2, zaś

64 i 128 — *missela* 3.

Nie zmienia się sposób liczenia odpowiednich wartości.

Liczby, które w opisany sposób utworzyliśmy, musimy teraz umieścić we właściwych miejscach pamięci. Przyjrzyjmy się następującemu programowi, umieszczającemu pokazany wyżej prostokąt w *playerze* 0.

```
10 REM DEFINIOWANIE RYSUNKU
   PLAYERA — PROSTOKĄT
11 REM MAMY NADZIEJĘ, ŻE
   PAMIĘĆ PLAYERA JEST
   „CZYSTA",
12 REM W PRZECIWNYM
   PRZYPADKU NALEŻY UMIEŚCIĆ
   W NIEJ STOSOWNĄ
13 REM LICZBĘ ZER
20 AD = PEEK (106)—16
30 POKE 106, AD—16:REM
   REZERWUJEMY PAMIĘĆ DLA
   PLAYERÓW
40 POKE 54279, AD:REM
   INFORMUJEMY SYSTEM GDZIE
   ZNAJDUJĄ SIĘ
   OPISY PLAYERÓW
41 REM
50 POKE 53277,3:REM „WŁĄCZAMY"
   GRAFIKĘ P—M
60 POKE 559,62: REM WYBIERAMY
   REZOLUCJE OBRAZU
70 START = AD 256 + 1024 + 100
80 FOR I = START TD START + 5:
   REM PĘTLA WPISUJĄCA DANE
90 READ B
100 POKE I,B
110 NEXT I
120 DATA 255,129,129,129,129,255
```

Wyjaśnienia wymagają instrukcje w następujących liniach:

50 — „włączenie" polega na wskazaniu systemowi, jakie elementy mają być uwidocznione na ekranie; wstawiając do komórki 53277 wartość 1, pozwalamy na przeniesienie na ekran obrazu misseli, — wartość 2 — obrazu playerów; w celu wykorzystania wszystkich możliwości, należy posłużyć się sumą tych wartości — stąd liczba 3.

70 — obliczanie adresu pierwszego bajtu, od którego należy rozpocząć wpisywanie definicji linii playera. Liczba 100 określa na ekranie linię, od której pokaże się nasz prostokąt. Poniżej przedstawione zostały numery linii zapewniające widzialność obrazka na monitorze. Podane liczby dotyczą rezolucji pojedynczej, w nawiasach znalazły się ich odpowiedniki dla rezolucji podwójnej.

Z powyższego rysunku wiemy, jakimi wartościami możemy opisywać poziome ustawienie *playera*. Zobaczymy teraz, jak to zrobić. Dla ułatwienia zapoznajmy się od razu z następującymi komórkami pamięci, które będą nam jeszcze potrzebne. Dla potrzeb naszego programu są to adresy dotyczące playera 0.

53248 — przechowuje wartość określającą poziome pozycje *playera*,

704 — definiuje jego kolor, jest to liczba postaci 16 (kod koloru) + (jasność)

53256 — opisuje jego szerokość, w sposób następujący:

0,2 — normalna, to znaczy jednemu bitowi z każdego bajtu *playera* odpowiada jeden punkt w linii ekranu, ma on szerokość 8 punktów (uff, to było trudne wyjaśnienie),

1 — podwójna, rozumieć należy w sposób oczywisty,

3 — poczwórna.

Natychmiast wykorzystajmy nowo zdobytą wiedzę do rozszerzenia programu o następne instrukcje:

```
130 POKE 53248,100:REM
    USTAWIENIE POZYCJI
    POZIOMEJ
140 POKE 704,64:REM NIECH
    BĘDZIE CZERWONY
150 POKE 53256,0:REM
    SZEROKOŚĆ NORMALNA
```

Dalej wzbogacić go możemy instrukcjami powodującymi poziomy przesuw, zgodny z ruchami joysticka. Oto one

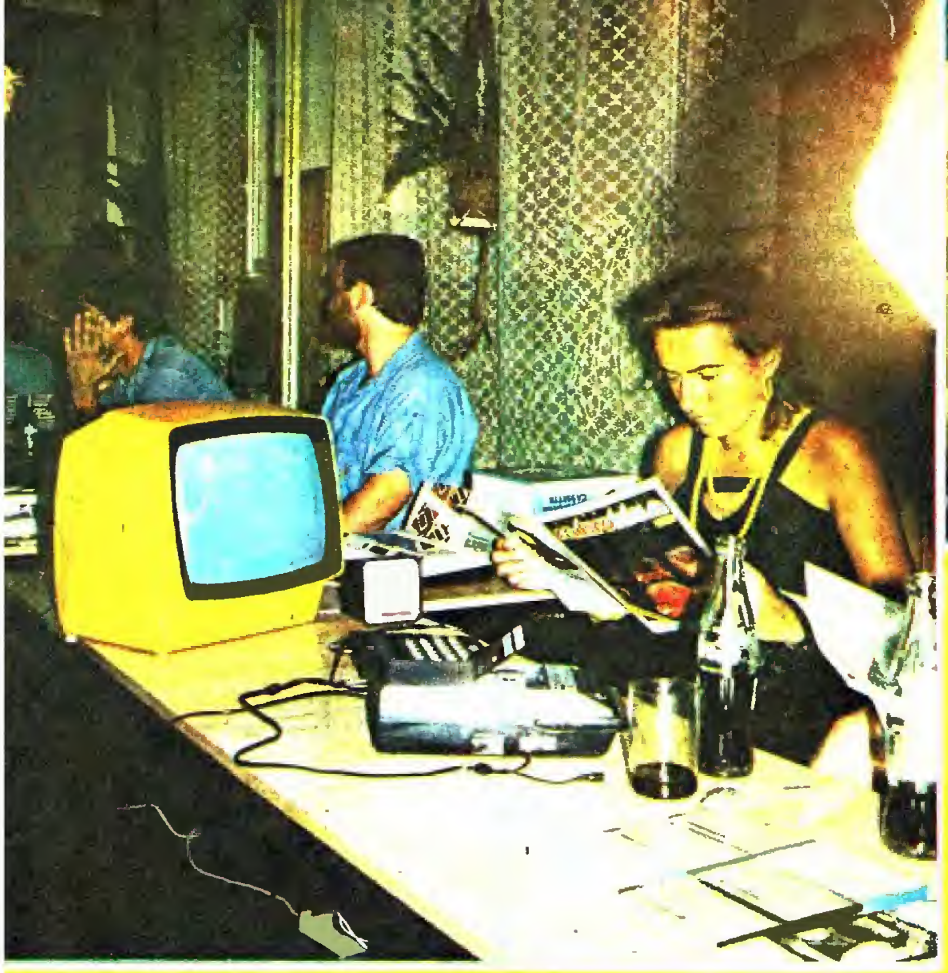
```
160 HPOZ = 100:REM USTALENIE
    POZYCJI POCZĄTKOWEJ
170 IF STICK(0) = 11 THEN
    HPOZ = HPOZ—1:REM W
    LEWO
180 IF STICK(0) = 7 THEN
    HPOZ = HPOZ + 1:REM W
    PRAWO
190 POKE 53248,HPOZ
200 GOTO 170
```

Kryją one jednak poważny błąd (oprócz tego, że tworzą nieskończoną pętlę). Proszę spróbować przesunąć prostokąt poza ekran (np. w prawo), zaobserwować efekt i wytłumaczyć go. Powinienem przy tym wytłumaczyć się, że nie zapomniałem o *misselach*, ani o pozostałych *playerach*. Po prostu uprościłem sobie problem, wiedząc, że operacje dotyczące wszystkich elementów grafiki P—M są jednakowe i przedstawiłem je na przykładzie *playera* 0. Jest to jednak moim obowiązkiem podać adresy wszystkich komórek, które mogą mieć znaczenie w trakcie programowania z wykorzystaniem przedstawionej techniki. Oto one:

704 kolor *playera* i *missela* 0,
705 kolor *playera* i *missela* 1,
706 kolor *playera* i *missela* 2,
707 kolor *playera* i *missela* 3,
53248 pozioma pozycja *playera* 0,
53249 pozioma pozycja *playera* 1,
53250 pozioma pozycja *playera* 2,
53251 pozioma pozycja *playera* 3,
53252 pozioma pozycja *missela* 0,
53253 pozioma pozycja *missela* 1,
53254 pozioma pozycja *missela* 2,
53255 pozioma pozycja *missela* 3,
53256 rozmiar *playera* 0,
53257 rozmiar *playera* 1,
53258 rozmiar *playera* 2,
53259 rozmiar *playera* 3,
53260 rozmiar wszystkich *misseli*, jest to liczba tworzona jako suma wartości opisujących szerokość każdego z nich.

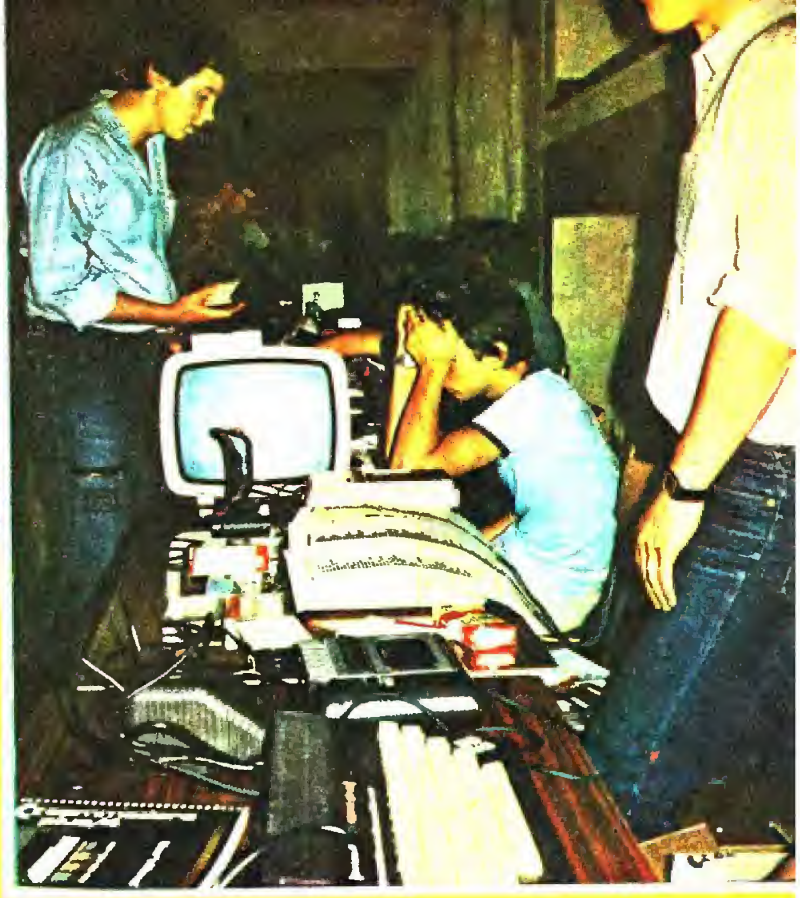
Na tym chciałbym zakończyć rozważania dotyczące playerów. Zachęcam wszystkich do wpisania tych kilku zaprezentowanych instrukcji i przeprowadzenia eksperymentów. Życzę przyjemnej zabawy i powodzenia przy tworzeniu swoich własnych ruszających się obrazków.

GRZEŚ



„SZKOŁA" E
KON

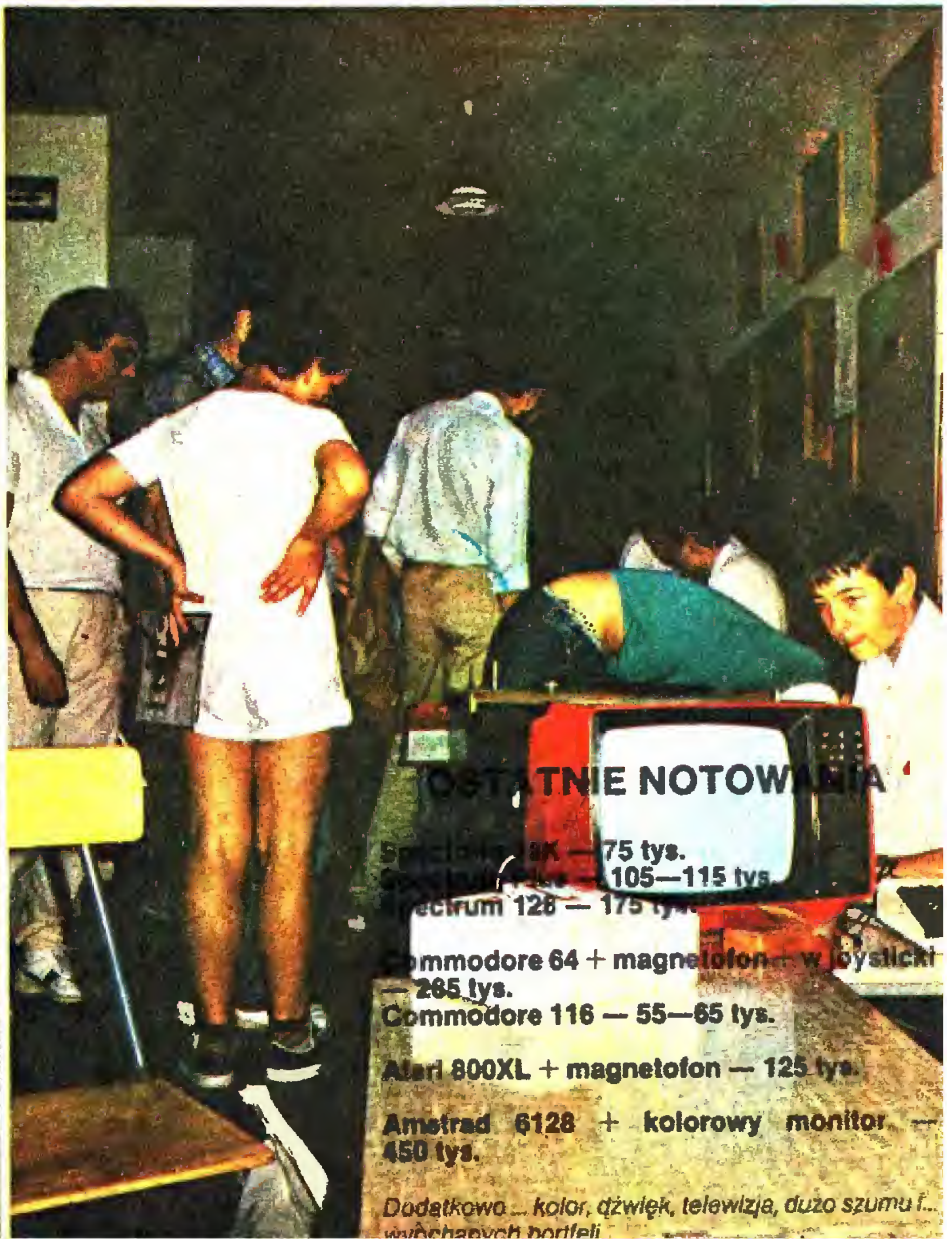
(czyli giełda n



DUKACJI...
MPUTEROWEJ

a Grzybowskiej)

Foto: Ryszard Rogoń



4

BASIC

(Podstawy programowania)

Kontynuujemy wykład z programowania w języku BASIC na mikrokomputer ZX Spectrum.

W wykładzie tym podamy podstawowe wiadomości i zasady tworzenia cykli obliczeniowych, zwanych także pętlami. Omówimy odpowiednie instrukcje BASIC-u, za pomocą których zapisujemy cykle obliczeniowe (pętle). Wiadomości te będą uzupełniane w następnych wykładach.

Wykład czwarty: cykle obliczenowe (pętle) zapisywanie pętli w programach w BASICu z wykorzystaniem — instrukcji **IF** — instrukcji cyklu **FOR I NEXT**.

W poprzednim wykładzie (wykład 3) zapoznaliśmy się między innymi z instrukcją IF. Stosowaliśmy ją np. w celu sprawdzenia odpowiedzi na pytanie: *„czy wprowadzamy dalej?"*. Jeżeli odpowiedzieliśmy *„tak"*, to wykonując instrukcję skoku bezwarunkowego (GO TO n) powracaliśmy do wykonania instrukcji o numerze n i powtarzaliśmy cykl obliczeń zapisany kilkoma instrukcjami. I tak np. w programie z przykładu 3.5 instrukcje o numerach 30—80 są wykonywane tyle razy, ile razy napisaliśmy *„tak"*.

Omówimy teraz zapisywanie za pomocą instrukcji IF cykli obliczeniowych zwanych także pętlami. Jak zwykle zaczynamy od przykładu.

Przykład 4.1

Napisać program na obliczenie sumy 5 liczb (dowolnych) wczytywanych w trakcie realizacji programu. Napisany program ma postać:

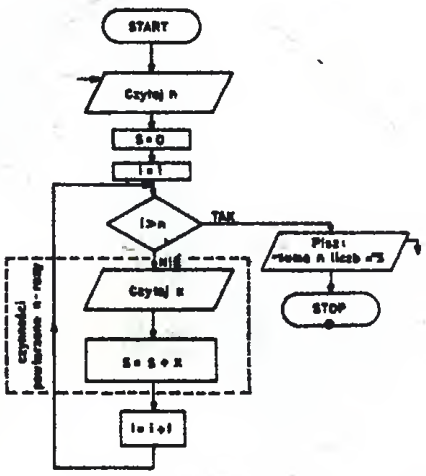
```
10 REM Przyklad 4.1
15 REM s-suma 5 liczb
       x-kolejna czytana liczba
20 LET s=0
25 INPUT "x=";x: PRINT x
30 LET s=s+x
35 INPUT "x=";x: PRINT x
40 LET s=s+x
45 INPUT "x=";x: PRINT x
50 LET s=s+x
55 INPUT "x=";x: PRINT x
60 LET s=s+x
65 INPUT "x=";x: PRINT x
70 LET s=s+x
75 PRINT "suma 5 liczb = ";s
```

Napisanie takiego programu, gdy chcemy zsumować więcej, np. 100 liczb, byłoby bezsensowne. Sumowanie dowolnej ilości liczb można przeprowadzić za pomocą programu, w którym występują instrukcje wielokrotnie wykonujące te same obliczenia dla różnych wartości danych.

Przykład 4.2

Napisać program obliczający sumę *n* liczb (dowolnych) wczytywanych w trakcie realizacji programu. W programie należy skorzystać z instrukcji **IF.**

a) schemat blokowy:



b) program

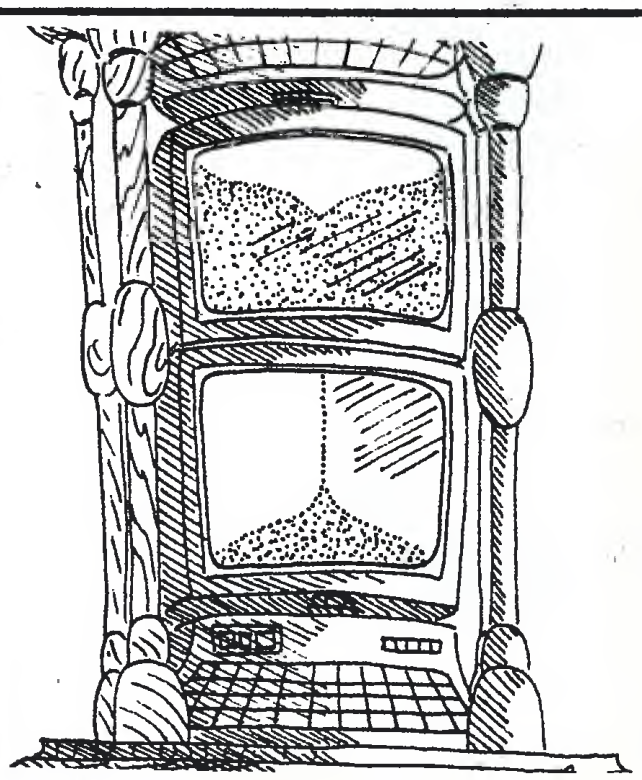
```
10 REM Przyklad 4.2
15 REM n-ilosc wczytywanych
       liczb
     x-kolejna czytana liczba
     i-licznik,s-suma n liczb
20 INPUT "n= ";n
25 PRINT "Obliczanie sumy ";n;
liczb"
30 PRINT '"wczytane liczby :"
35 LET s=0
40 LET i=1
45 IF i>n THEN GO TO 80
50 INPUT x
55 PRINT x;"  ";
60 LET s=s+x
65 LET i=i+1
70 GO TO 45
80 PRINT '"suma ";n;" liczb =
";s
```

c) wyniki dla przykładowych danych

```
Obliczanie sumy 5 liczb

wczytane liczby :
2  3  5  6  8

suma 5 liczb = 24
```

Zauważmy, że jeżeli rozwiązywanie problemu zawiera pewien proces powtarzalny, to w schemacie blokowym pojawia się pętla (cykle obliczeniowe). Pętla musi zawierać określenie sposobu jej zakończenia. Czynność polegająca na wykonaniu każdej operacji w jednym przebiegu pętli nazywa się cyklem pętli.

Zajmiemy się teraz analizą schematu blokowego z przykładu 4.2. Czynności opisane blokami: czytanie *x* i sumowanie ($S = s + x$) mają się wykonać *n*-razy dla różnych wartości *x* (w tym miejscu wartość *n* jest już znana). Wykonanie tych czynności dla jednej wartości *x* to cykl pętli.



Rys. Michał Przybyłowski

Zapis w tym bloku odczytujemy w następujący sposób: jeżeli warunek $i > n$ jest spełniony, to przejdź do wypisania wartości sumy n-liczb; jeżeli jest nie spełniony, to powtórz obliczenia tzn.:

— czytaj x
— dodaj x do sumy ($s = s + x$)
— dodaj do i liczbę 1; obliczoną wartość podstaw pod zmienną i ($i = i + 1$)
— powróć do pytania $i > n$

W tym przykładzie zmienna i służy do zliczania ilości liczb, a zarazem zlicza liczbę wykonanych cykli w pętli, można także powiedzieć, że zmienna i odgrywa również rolę licznika „obrotów" pętli; jej wartość zmienia się o stałą liczbę równą 1. Po zakończeniu obliczeń, tzn. w razie spełnienia nierówności $i > n$, wartość zmiennej i wynosi $n + 1$.

Zapiszemy teraz program z przykładu 4.2 w postaci:

```
10 REM Przyklad 4.2.1
15 REM n-ilosc wczytywanych
       liczb
     x-kolejna czytana liczba
     s-suma n liczb
20 INPUT "n= ";n
25 PRINT "Obliczanie sumy ";n;
liczb"
30 PRINT "wczytane liczby :"
35 LET s=0
40 FOR i=1 TO n STEP 1
45 REM i-zmienna sterujaca
50 INPUT X
55 PRINT X;"  ";
60 LET s=s+X
70 NEXT i
80 PRINT "suma ";n;" liczb =
";s
```

Wprowadźmy i wykonajmy ten program dla tych samych liczb co program z przykładu 4.2. Otrzymane wyniki są takie same jak poprzednio.

Do zapisu pętli w BASICu przeznaczone są dwie specjalne instrukcje zwane instrukcjami cyklu (lub instrukcjami powtórzeń), są to instrukcje czynne FOR i NEXT. Są one zawsze używane razem i mają dwie postacie ogólne:

n_1 FOR $z_p = w_1$ TO w_2 STEP w_3
⋮
n_2 NEXT z_p

lub

n_1 FOR $z_p = w_1$ TO w_2
⋮
n_2 NEXT z_p

gdzie: n_1, n_2 — numery wierszy
z_p — zmienna sterująca (lub zmienna kontrolowana, lub wskaźnik pętli) jest zmienną prostą numeryczną, nazwa zmiennej sterującej musi być jednoliterowa,
w_1 — wyrażenie arytmetyczne określające wartość początkową zmiennej sterującej,
w_2 — wyrażenie arytmetyczne określające wartość końcową zmiennej sterującej,
w_3 — wyrażenie arytmetyczne określające wartość kroku zmiany zmiennej sterującej.
w_1, w_2, w_3 — nazywane są parametraml cyklu.

Jeżeli krok zmiany zmiennej sterującej ma wartość 1 ($w_3 = 1$) to fragment STEP w_3 może być pominięty. Instrukcja FOR przyjmuje wtedy postać drugą.

Przykłady instrukcji FOR:

a) 100 FOR $i = 1$ TO n
⋮
140 NEXT i

b) 60 FOR $k = i + k$ TO $5 * i + 2 * k + 1$ STEP 2
⋮
160 NEXT k

c) 70 FOR j =—25 TO 20
170 NEXT j

d) 50 FOR $x = x1$ TO $x2$ STEP $x3$
100 NEXT x

e) 100 FOR $i = 1$ TO n STEP k
150 NEXT i

W przykładach b, d, e, przed wejściem do pętli muszą być znane wartości i, $x1$, $x2$, $x3$, oraz l, n, k.

Instrukcje cyklu umożliwiają proste zapisanie w programie tej części schematu blokowego, którą opisywaliśmy za pomocą instrukcji warunkowej IF. W naszym przykładzie zamiast instrukcji:

40 LET i = 1
45 IF i > n THEN GO TO 80

oraz

65 LET $i = i + 1$
70 GO TO 45

napisaliśmy instrukcje:

40 FOR i = 1 TOW STEP 1

oraz

70 NEXT i

Przy tworzeniu pętli można wyróżnić pięć grup czynności, z których każda została opisana w postaci odpowiedniego bloku na powyższym schemacie.

Wyjaśnień dodatkowych wymaga blok 1.

Parametry cyklu w_1, w_2, w_3 są wyrażeniami arytmetycznymi, a więc mogą być to stałe numeryczne nazwy zmiennych numerycznych lub wyrażenia arytmetyczne. Przed wejściem do pętli muszą być nadane wartości zmiennym numerycznym wchodzącym w skład wyrażeń arytmetycznych, reprezentujących parametry cyklu w_1, w_2, w_3.

Zapamiętajmy pewne reguły obowiązujące przy pisaniu programów z instrukcjami FOR i NEXT:

— instrukcję postaci:
n FOR $z_p = w_1$ TO w_2 STEP w_3
piszemy w miejscu, które odpowiada na schemacie blokom o numerach 2 i 3 (uprzednio należy ustalić wartości wyrażeń w_1, w_2, w_3, blok numer 1),
— instrukcję postaci:
n_1 NEXT z_p
piszemy w miejscu, które odpowiada na schemacie blokowi o numerze 5.
— blok o numerze 4, to ciąg instrukcji, które mają być wykonane tyle razy, ile wskazują parametry cyklu.

Tak więc realizację podanego schematu można w BASICu zapisać ciągiem instrukcji postaci;

```
40 REM instrukcje przed petla,
       blok P1
50 REM nadanie wartosci zmien-
       nym wystepujacym w para
       metrach cyklu (jezeli
       wystepuja)
60 FOR z=w1 TO w2 STEP w3
70 REM ciag instrukcji -
80 REM wnetrze petli
90 NEXT z
100 REM instrukcje po petli,
       blok P2
```

Ten sam ciąg instrukcji zapisany w BASICu za pomocą instrukcji IF wygląda następująco:

```
 40 REM instrukcje przed petla,
        blok P1
 50 REM nadanie wartosci zmien-
        nym wystepujacym w para
        metrach cyklu (jezeli
        wystepuja)
 55 REM G, H - zmienne dostepne
        jedynie translatorowi
        jezyka BASIC
 60 LET z=w1
 62 LET G=w2
 64 LET H=w3
 66 IF SGN (H)*(z-G)>0 THEN GO
TO 100
 70 REM ciag instrukcji -
 80 REM wnetrze petli
 90 LET z=z+H
 92 GO TO 66
100 REM instrukcje po petli,
        blok P2
```

dokończenie na str. 20

dokończenie ze str. 19

Z podanego ciągu instrukcji wynika, że na wykonanie instrukcji cyklu FOR—NEXT składają się następujące czynności:

1. obliczenie wartości wyrażenia w_1 i nadanie zmiennej sterującej z wartości początkowej (wiersz 60),
2. obliczenie wartości wyrażeń w_2 i w_3 (z uwzględnieniem nowej wartości zmiennej) oraz zapamiętanie wartości końcowej w_2 i kroku w_3 pod zmiennymi G i H niedostępnymi dla programu użytkownika (wiersz 62 i 64),
3. sprawdzenie warunku niewykonywania cyklu (wiersz 66), przy spełnieniu tego warunku następuje przejście do instrukcji występującej bezpośrednio po instrukcji NEXT.(SGN *(x)*) jest to funkcja standardowa — podaje ona znak zmiennej x, wartości wykonania tej funkcji wynoszą (—1) jeżeli $x < 0$; 0 jeżeli $x = 0$; 1 jeżeli $x > 0$),
4. wykonanie ciągu instrukcji będącego wnętrzem pętli,
5. nadanie nowej wartości zmiennej sterującej (wiersz 90),
6. przejście do wykonywania następnego cyklu (wiersz 92).

Z podanych czynności wynika następujące działanie instrukcji cyklu:

— jeżeli wartość kroku jest większa od zera ($w_3 > 0$), to wartość początkowa w_1 musi być nie większa od wartości końcowej w_2 ($w_1 \leq w_2$),

— jeżeli wartość kroku jest mniejsza od zera ($w_3 < 0$) to wartość początkowa w_1 musi być mniejsza od wartości końcowej w_2 ($w_1 \geq w_2$)

W przeciwnym przypadku cykl nie jest wykonywany, a zmienna sterująca przyjmuje wartość początkową. Po wykonaniu instrukcji cyklu wartość zmiennej sterującej jest większa od w_2 o krok w_3 tzn. $z = w_2 + w_3$ (patrz: wydruk do przykładu 4.3).

Podaliśmy obszerną porcję informacji teoretycznych, postaramy się je zapamiętać, wykonując kilka przykładów

Przykład 4.3

Danych jest 15 liczb. Należy wczytywać po jednej liczbie w czasie realizacji programu oraz drukować tę liczbę, jej kwadrat i sześcian.

Program jest prosty i podajemy go bez schematu blokowego.

```
 10 REM Przyklad 4.3
 20 REM x-czytana liczba
 30 PRINT "     x     ";"     x↑2
";"     x↑3     "
 40 FOR i=1 TO 15 STEP 1
 50 INPUT "x= ";x
 60 PRINT "     ";x;"     ";"     "
;x*x;"     ";"     ";x*x*x;"     "
 65 REM liczby moga byc dowolne
       ujemne,rowne zero lub
       dodatnie,a wiec nie moze
       my zastosowac operatora
       potegowania ( ↑ )

 70 NEXT i
 80 PRINT "i=";i
```

Wprowadzmy i wykonajmy ten program. Niżej prezentujemy otrzymane wyniki dla przykładowych danych:

```
x          x↑2        x↑3
-1          1         -1
-2          4         -8
-3          9        -27
i =16
```

```
0          0
4          8
9         27
16        64
25       125
36       216
49       343
64       512
81       729
36       216
25       125
16        64
```

Patrząc na programy z przykładów 4.2.1 i 4.3 **zauważamy, że instrukcje cyklu zostały zastosowane do zliczenia powtórzeń wykonywania ciągu instrukcji.** Można powiedzieć, że jest to podstawowa funkcja tych instrukcji.

W przykładzie 4.2 instrukcje cyklu spowodują wykonanie ciągu instrukcji (wiersze 50, 55, 60) *n*-razy, (wartość *n* jest już w tym momencie znana), a w przykładzie 4.3 — wykonanie ciągu instrukcji (wiersze 50, 60) — 15 razy.

Przykład 4.4

Napisać program na mnożenie danej liczby *x* przez 10 kolejnych liczb naturalnych, poczynając od zadanej liczby *l*.

Program jest prosty i podajemy go bez schematu blokowego.

```
10 REM Przyklad 4.4
20 INPUT "x= ";x
30 INPUT "l= ";l
40 FOR j=l TO l+10-1 STEP 1
50 PRINT x;" * ";j;" = ";j*x
60 NEXT j
```

Wprowadźmy i wykonajmy ten program. Niżej prezentujemy otrzymane wyniki dla przykładowych danych.

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

W programie tym ciąg instrukcji powtarzanych (wnętrze pętli) składa się z jednej instrukcji:

70 PRINT x; "*" ; *j* ; "=" ; *j* * *x*

W instrukcji tej występuje zmienna *j*, która jest zmienną sterującą instrukcji cyklu napisanej w wierszu 40. Wykorzystujemy tu jej wartość. **W ciągu instrukcji powtarzanych (we wnętrzu pętli) można wykorzystywać wartość zmiennej sterującej, ale nie należy zmieniać jej wartości.**

We wnętrzu pętli można zmieniać (ale się nie zaleca) wartość zmiennej sterującej. Nowa wartość nadana tej zmiennej zostanie uwzględniona przy kolejnym obliczaniu nowej wartości zmiennej sterującej i sprawdzaniu z wartością końcową.

Przykład 4.5

Dany jest program. W programie tym w linii 70 następuje zwiększenie wartości zmiennej sterującej o 3:

```
40 REM Przyklad 4.5
50 FOR i=1 TO 18
60 PRINT i,
70 LET i=i+3
80 PRINT i
90 NEXT i
```

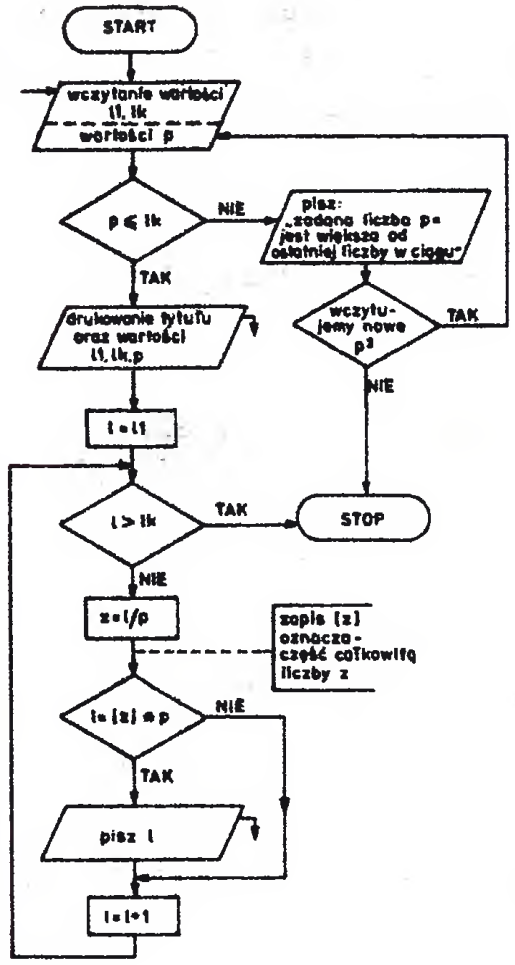
Instrukcje wnętrza pętli wykonane zostaną tylko 5 razy zamiast 18. Wprowadźmy ten program i wykonajmy. Sprawdźmy wynik z wynikiem podanym niżej.

```
1          4
5          8
9         12
13        16
17        20
```

Przykład 4.6

Dany jest ciąg liczb naturalnych $l_1, l_2 \ldots l_k$. Napisać program na wybieranie spośród tych liczb i drukowanie liczb podzielnych (bez reszty) przez zadaną liczbę p (p-naturalne, o wartości nie większej od wartości maksymalnej liczby w ciągu tzn. $p \leq l_k$).

a) schemat blokowy



b) program

```
  10 REM Przyklad 4.6
  15 PRINT "wybieranie z danego
ciagu liczb naturalnych liczb po
dzielnych   przez zadana liczbe
p"
  20 REM l1-pierwsza liczba w
         ciagu
         lk-ostatnia liczba w
         ciagu
         p-zadana liczba
  25 INPUT "l1= ";l1,"lk= ";lk
  30 INPUT "p= ";p
  40 IF p<=lk THEN GO TO 100
  50 PRINT '"podana liczba p= ";
p'"jest wieksza od ostatniej
   liczby w ciagu = ";lk
  60 PRINT '"wczytujemy nowe p(t
ak/nie)?";
  70 INPUT x$: PRINT x$
  80 IF x$="tak" THEN GO TO 30
  90 STOP
 100 PRINT '"liczby podzielne pr
zez ";p'"w ciagu ";l1;".....";l
k
 105 PRINT '
 110 FOR l=l1 TO lk STEP 1
 120 LET z=l/p
 130 IF l=INT (z)*p THEN PRINT l
" ";
 140 NEXT l
 150 STOP
```

c) wyniki dla przykładowych danych

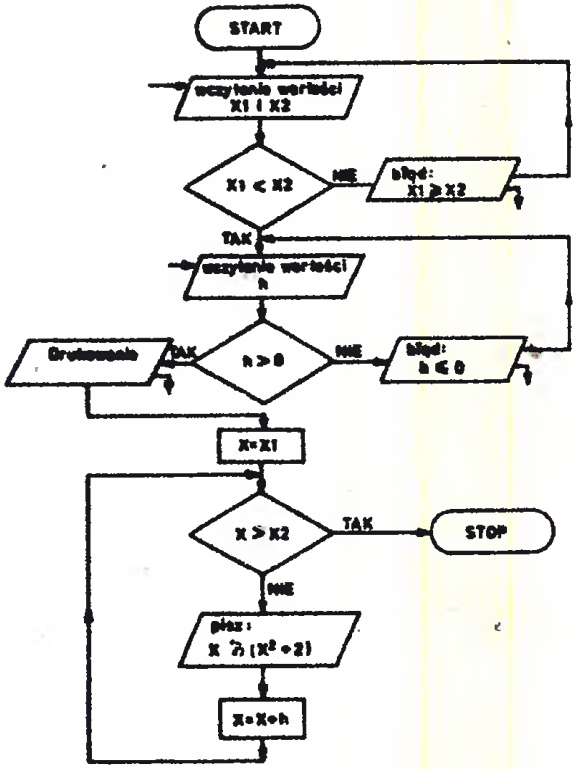
```
wybieranie z danego ciagu liczb
naturalnych liczb podzielnych
przez zadana liczbe p

liczby podzielne przez 3
w ciagu 6.....30

6  9  12  15  18  21  24  27  30
```

Przykład 4.7

Napisać program drukowania tablicy wartości funkcji $ln\ (x^2+2)$ dla argumentu x zmieniającego się w granicach od $x1$ do $x2$ z krokiem h. Wartości $x1$, $x2$, h wczytywane są w czasie realizacji programu. Należy sprawdzić czy spełniają one warunki:

$$x1 < x2 \text{ i } h > 0.$$

a) schemat blokowy




```
  10 REM Przyklad 4.7
  15 PRINT "Tablicowanie funkcji
 LN(x↑2+2)  w przedziale <x1,x2>
 z krokiem h"
  20 INPUT "x1= ";x1,"x2= ";x2
  30 IF x1<x2 THEN GO TO 60
  40 PRINT '"zle wartosci x1 i x
2 - x1>=x2"'"x1 = ";x1,"x2 = ";x
2
  50 GO TO 20
  60 INPUT "h= ";h
  70 IF h>0 THEN GO TO 95
  80 PRINT '"zla wartosc kroku h
 - h<=0"'"h = ";h
  90 GO TO 60
  95 PRINT '"Dane: ";"x1 = ";x1;
"   x2 = ";x2;"   h = ";h
 100 PRINT '"    x","  Ln(x↑2+2)"
 110 FOR x=x1 TO x2 STEP h
 120 PRINT x,LN (x*x+2)
 130 NEXT x
 140 STOP
```

dokończenie na str. 22

dokończenie ze str. 21

c) wyniki dla przykładowych danych

```
Tablicowanie funkcji LN(x↑2+2)
w przedziale <x1,x2> z krokiem h

Dane: x1 = 1    x2 = 5   h = 1

    x                 Ln(x↑2+2)
1                     1.0986123
2                     1.7917595
3                     2.3978953
4                     2.8903718
5                     3.2958369
```

oraz

```
Tablicowanie funkcji LN(x↑2+2)
w przedziale <x1,x2> z krokiem h

zle wartosci x1 i x2 - x1>=x2
x1 = 5                x2 = 1

zla wartosc kroku h - h<=0
h = -1

Dane: x1 = 1    x2 = 5   h = 1

    x                 Ln(x↑2+2)
1                     1.0986123
2                     1.7917595
3                     2.3978953
4                     2.8903718
5                     3.2958369
```

Przykład 4.8

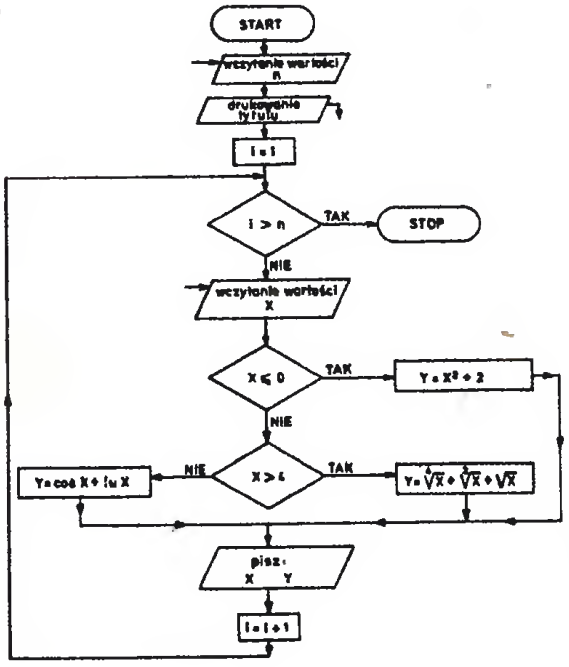
Napisać program drukowania tablicy wartości funkcji y zadanej wzorem:

$$y = \begin{cases} x^2 + 2 & \text{dla } x \le 0 \\ \cos(x) + \ln(x) & \text{dla } 0 < x \le 4 \\ (x)^{1/4} + (x)^{1/3} + (x)^{1/2} & \text{dla } x > 4 \end{cases}$$

dla *n* — wartości zmiennej *x*

Kolejne wartości zmiennej x mają być wczytywane w czasie realizacji programu.

a) schemat blokowy



b) program

```
  10 REM Przyklad 4.8
  15 PRINT "tablicowanie funkcji
 y dla-        n-wartosci zmienne
j x"
  20 INPUT "n = ";n
  30 PRINT "     x","     y"
  35 PRINT
  40 FOR i=1 TO n
  50 INPUT "x = ";x
  60 IF x<=0 THEN LET y=x*x+2: G
O TO 90
  70 IF x>4 THEN LET y=SQR (SQR
(x))+x↑(1/3)+SQR (x): GO TO 90
  80 LET y=COS x+LN x
  90 PRINT x,y
 100 NEXT i
 110 STOP
```

c) wyniki dla przykładowych danych

```
tablicowanie funkcji y dla
  n-wartosci zmiennej x

     x                    y

-2                    8
-1                    3
1.5                   0.47620231
3                     0.10861979
16                    6.5198421
```

Na tym kończymy omawianie podstawowych informacji o pętlach (cyklach obliczeniowych) oraz o instrukcjach cyklu FOR i NEXT. Wiadomości te będą uzupełniane i rozszerzane w następnych wykładach.

Zadania

4.1. Podać ile razy i jakie wartości zostaną wprowadzone w trakcie wykonywania następujących fragmentów programów:

```
10 REM ZAD.4.1.1
20 FOR x=-21 TO -1 STEP 3
30 PRINT "x = ";x+1
40 NEXT x
```

```
10 REM ZAD.4.1.2
20 FOR i=0 TO 5
30 LET x=-3+i*0.25
40 PRINT "x = ";x
50 NEXT i
```

```
10 REM ZAD.4.1.3
20 LET b=32
30 FOR j=1 TO b STEP 4
40 PRINT "j =";j
50 LET j=j+2
60 LET b=b+2
70 NEXT j
80 PRINT "b = ";b
```

```
10 REM ZAD.4.1.4
20 LET i=5
30 FOR i=i-10 TO i+10 STEP ABS
(i/2)
40 PRINT "i = ";i
50 NEXT i
60 PRINT "i = ";i
```

4.2. Napisać następujące programy, tak, aby nie występowały w nich instrukcje cyklu FOR i NEXT — wykorzystać instrukcje IF:

4.2.1. program z **przykładu 4.7**

4.2.2 program z **przykładu 4.8**

Z.Z.

4

LOGO

Program w języku Logo składa się z procedur — tyle wiemy już z dotychczasowych rozważań. Wiemy też, że procedury mogą być wywoływane od różnych parametrów, a zatem wywołane w odmienny sposób, mogą obliczać różne wartości — dla innych danych, albo też, jak to się niebawem okaże, wykonywać zupełnie inne obliczenia dla różnych wartości parametrów. Zazwyczaj rozwiązanie problemu wymaga napisania więcej niż jednej procedury. Procedury te muszą mieć możliwość porozumiewania się między sobą, przekazywania sobie nawzajem wyników obliczeń. W tym celu w Logo przyjęto, że każda procedura może zwracać wartość. Wewnątrz treści procedury należy umieścić słowo WYNIK, a następnie wielkość, która powinna być dostępna na zewnątrz procedury. Wielkością tą może być dowolny obiekt Logo, może też być to dowolne wyrażenie, które po obliczeniu „stanie się" obiektem (wyrażenie jest po prostu skomplikowaną nazwą obiektu). Wykonanie komendy WYNIK powoduje, że nazwie procedury nadawana jest wartość i wykonywanie tej procedury jest przerywane. Nazwą takiej procedury posługujemy się tak jak obiektem, który jest jej wartością. Oto proste przykłady:
1) Obliczamy pole i objętość kuli o promieniu r, korzystając z procedury liczącej pole koła o tym samym promieniu (zauważmy, że: $\frac{4}{3}\pi r^3 = \frac{4\pi r^2 r}{3}$):

```
?
OTO KOŁO :r
WYNIK PI↑ * :r * :r
JUŻ

OTO KULA :r
PISZ [pole kuli =]
PISZ 4 * KOŁO :r
PISZ [objętość kuli =]
PISZ 4 / 3 * :r * KOŁO :r
JUŻ
```

2) Rysujemy wielokąt foremny, korzystając z procedury liczącej, o jaki kąt powinien obrócić się żółw zależnie od liczby boków wielokąta:

```
?
OTO WIELOKĄT :lb :db
POWTÓRZ :lb [np :db lewo ilestop
ni :lb]
JUŻ

OTO ILESTOPNI :lb
WYNIK 360 / :lb
JUŻ
```

Napiszmy teraz procedurę „wielokąt 2" tak:

```
?
OTO SPIRALA :odl :kąt :skok
NP :odl
LW :kąt
SPIRALA :odl + :skok :kąt :skok
JUŻ
```

i wywołajmy ją. Na ekranie nie będzie żadnego rysunku, ale Logo zapyta, co zrobić z wynikiem zwracanym przez procedurę „wielokąt". Komenda WYNIK przekazuje dane na wyższy poziom i jednocześnie zatrzymuje działanie procedury. Jest to logicznie uzasadnione — po co jeszcze coś obliczać skoro wynik jest już ustalony. Gdybyśmy w ostatniej procedurze „wielokąt" zamiast WYNIK napisali STOP, nie otrzymalibyśmy na ekranie w ogóle żadnego efektu — procedura nic nie robi i nie ma żadnej wartości. Komenda STOP po prostu kończy, działanie procedury. Na obecnym etapie znajomości Logo może się wydawać, że używanie STOP jest pozbawione większego sensu. Tymczasem okaże się później, że komenda ta jest bardzo potrzebna i często stosowana. W angielskim Logo WYNIK to OUTPUT, a komenda STOP brzmi tak samo po polsku.

Procedury rozwiązujące ten sam problem komunikują się między sobą, wywołują się nawzajem. Gdy jakąś czynność należy powtórzyć wielokrotnie, najprościej jest, gdy procedura, która tę czynność wykonuje, wywołała samą siebie. Jest to tak zwane wywołanie rekurencyjne. Logo pozwala na posługiwanie się rekursją w programowaniu. Co więcej, w Logo rekursja jest jednym z podstawowych narzędzi programowania. Dla niektórych obliczeń program rekurencyjny jest najprostszym i najłatwiej narzucającym się rozwiązaniem. Tak jest choćby przy liczeniu „silni" liczby naturalnej:

```
?
OTO SILNIA :n
JEŚLI :n = 0 [wy 1]
WY :n * SILNIA :n - 1
JUŻ
```

W procedurze tej pojawiła się nader użyteczna instrukcja warunkowa JEŚLI. Jej zastosowanie w tym programie jest oczywiste, szczegóły JEŚLI nieco niżej. Tymczasem wróćmy do rekursji. Służy ona w Logo dosłownie do wszystkiego. Korzystając z niej można szybko i elegancko programować. Spróbujmy natomiast napisać jakikolwiek większy program, zupełnie nie posługując się rekursją. Jestem pewny, że będzie to niełatwe zadanie, jeśli w ogóle wykonalne. Logo dysponuje tylko jedną instrukcją iteracyjną (powtórzenie) — jest to już omawiana instrukcja POWTÓRZ. Wiemy już, że nie daje ona zbyt wielkich możliwości — trzeba z góry określić, ile razy należy wykonać podane po słowie POWTÓRZ instrukcje. W Logo nie ma instrukcji skoku w rodzaju choćby Basic'owego GO TO. Zatem zakładanie pętli z użyciem skoku jest niemożliwe. Pozostaje rekursja, z której należy jak najwięcej korzystać. Przy pewnej wprawie programowanie rekurencyjne jest bardzo naturalne i oczywiste. Trzeba się tylko przestawić na ten sposób myślenia. Łatwiej z pewnością będzie tym, dla których Logo jest pierwszym poznawanym językiem programowania, trudniej tym, którzy przyzwyczaili się do programowania w języku bez rekursji np. w Basic'u. Mają oni już utrwalone schematy myślenia programistycznego i trudniej im przyjdzie znajdować rozwiązania rekurencyjne, bo mimo woli najpierw będą się głowić nad rozwiązaniem iteracyjnym (za pomocą pętli) danego problemu. Tacy fani programowania najlepiej zrozumieją to, co fachowcy wciąż powtarzają: że ważne jest nie tylko to, by jak najszybciej zetknąć się z komputerem i programowaniem, ale także, a może nawet głównie to, by zrobić to w sposób właściwy — tak, aby od początku „zakodować" sobie odpowiednie odruchy myślenia programistycznego. A oto przykład nadzwyczaj prostej a dającej niespodziewanie ciekawe efekty procedury rekurencyjnej:

```
?
OTO WIELOKĄT2 :lb :db
WYNIK ILESTOPNI :lb
POWTÓRZ :lb [np :db lw ilestopni
 :lb]
JUŻ
```

Jak wskazuje to jej nazwa, procedura rysuje spiralę — jej kształt jest zależny od podanych parametrów: długości dodawanego odcinka i kąta, o jaki za każdym razem obraca się żółw. Czytelnikom siedzącym przy komputerze radzę wypróbować działanie procedury „spirala" dla różnych wartości parametrów. Prawda, że rysunki są interesujące? Ci, którzy testowali procedurę zauważyli już, że procedura „spirala" raz uruchomiona działa bez końca. Trzeba ją

dokończenie na str. 24

dokończenie ze str. 23

przerywać naciskając klawisz BREAK. Aby procedura kończyła swe działanie bez brutalnej interwencji z zewnątrz, trzeba w jej wnętrzu umieścić instrukcję stopu, obwarowaną jakimś sensownym warunkiem. Instrukcja warunkowa w polskim Logo zaczyna się od słowa JEŚLI, dalej następuje warunek (na przykład: n=0) i dwie listy. Pierwsza z nich zawiera instrukcje, które są wykonywane, gdy warunek jest spełniony, druga — instrukcje do wykonania przy niespełnionym warunku. Pierwsza lista musi być podana, drugą można pominąć. Instrukcja warunkowa w procedurze „silnia" jest zapisana właśnie z pominięciem drugiej listy. W tej sytuacji Logo stwierdzając, że zadany warunek nie jest spełniony, po prostu przechodzi do wykonywania następnej po JEŚLI instrukcji. Zauważmy, że procedurę „silnia 2" możemy równoważnie zapisać tak:

```
?
OTO SILNIA2 :n
JEŚLI :n = 0 [wy 1] [wy :n * sil
nia2 :n - 1]
JUŻ
```

W procedurze „spirala" instrukcję warunkową wykorzystamy dla zatrzymania rysowania spirali w odpowiednim momencie. Przyjmijmy, że na przykład procedura ma przerwać rysowanie, gdy jednorazowo kreślony odcinek będzie miał długość większą niż 100:

```
OTO SPIRALA2 :odl :kąt :skok
JEŚLI :odl > 100 [stop]
NP :odl
LW :kąt
SPIRALA2 :odl + :skok :kąt :sko
JUŻ
```

W angielskim Logo zamiast JEŚLI należy pisać IF. Inne zasady dotyczące zapisu instrukcji warunkowej są takie same, jak w polskiej wersji języka. A oto jeszcze jeden program w istotny sposób wykorzystujący instrukcję warunkową i możliwość przekazywania wyników obliczeń przez procedury. Podany niżej komplet procedur oblicza i wypisuje na ekranie pierwiastki równania kwadratowego oraz współrzędne wierzchołka paraboli, będącej wykresem funkcji kwadratowej o zadanych współczynnikach. Program doskonale nadaje się do weryfikacji rozwiązań szkolnych zadań „do odrobienia" w domu!

```
?
OTO RÓWKWAD :a :b :c
MIEJSCAZERowe :a :b :c
WIERZCHOŁEK :a :b :c
JUŻ

OTO MIEJSCAZEROWE :a :b :c
JEŚLI 0 > DELTA :a :b :c [stop]
JEŚLI 0 = DELTA :a :b :c [pisz [
X0 =] PISZ - :b / ( 2 * :a ) STO
P]
PISZ [x1 =] PISZ X1 :a :b :c
PISZ [x2 =] PISZ X2 :a :b :c
JUŻ

OTO DELTA :a :b :c
WYNIK :b * :b - 4 * :a * :c
JUŻ

OTO X1 :a :b :c
WY ( - :b - PIERWIASTEK DELTA :a
:b :c ) / ( 2 * :a )
JUŻ

OTO X2 :a :b :c
WY ( - :b + PKW DELTA :a :b :c )
/ ( 2 * :a )
JUŻ

OTO WIERZCHOŁEK :a :b :c
PISZ [współrzędne wierzchołka]
PISZ [x =]
PISZ - :b / ( 2 * :a )
PISZ [y =]
PISZ ( - DELTA :a :b :c ) / ( 4
* :a )
JUŻ
```

Nic dotychczas nie mówiliśmy o zmiennych w Logo. A zatem zmienną w tym języku można utożsamiać z nazwą. Jest to jak gdyby etykietka przyporządkowana obiektowi Logo, a nie tak jak w niektórych innych językach pudełko, do którego wkłada się wartości zmiennej. Obiekty opatrywane są etykietkami za pomocą instrukcji przypisania. Po słowie PRZYPISZ należy podać nazwę zmiennej, poprzedzoną cudzysłowem, a następnie wartość nadawaną tej zmiennej. Wartość ta powinna mieć normalną dla Logo postać: liczby podaje się zwyczajnie, wartości zmiennych powinny być poprzedzone dwukropkiem (zapis taki jak dla parametrów procedur), nazwy funkcji (procedur zwracających wynik) wymienione normalnie, listy jako ciągi obiektów ujęte w nawiasy kwadratowe (o listach i operacjach na nich będzie szczegółowo w następnych odcinkach), słowa (napisy) poprzedzone cudzysłowem. Na przykład:
PRZYPISZ "y 89
PRZYPISZ "ala "kot
PRZYPISZ "x :y — etykietka dla x została przypisana także wartości 89. Przypisywać można także skomplikowane wyrażenia np:
PRZYPISZ "X1 (—:b — PIERWIASTEK delta :a :b :c) /2 + :a

W angielskim Logo odpowiednikiem słowa PRZYPISZ jest MAKE. Instrukcję przypisywania można wykorzystać choćby do zaprogramowania iteracyjnej wersji procedury „silnia":

```
?
OTO SILNIA3 :n
PRZYPISZ "m 1
PRZYP "i 0
POWTÓRZ :n [przyp "i :i + 1 przy
p "m :m * :i]
WY :m
JUŻ
```

Na zakończenie, jak zwykle, przypomnijmy sobie nowe słowa poznane w tym odcinku. A więc:

PRZYPISZ (PRZYP) przypisuje wartości nazwę: np. przyp "a11 to nadanie nazwy a liczbie 11 (ang. MAKE)

WYNIK (WY) n przypisuje procedurze wartość i kończy jej działanie (ang. OUTPUT (OP))

STOP kończy działanie procedury (przerywa — powoduje „wyjście" z procedury) (ang. STOP)

Jeśli war akcja1 akcja2 jeśli warunek jest prawdą wykonuje się lista — akcja1, w przeciwnym przypadku akcja 2 (może jej zresztą nie być)

PIERWIASTEK (PKW) n daje (zwraca) pierwiastek kwadratowy n (ang.SQRT)

SIS

W bardzo wielu zastosowaniach matematyki korzysta się z funkcji zwanych wielomianami. Wynika to ze szczególnych własności tych funkcji:

— łatwość obliczenia wartości wielomianu za pomocą skończonej liczby mnożeń i dodawań;
— łatwość wykonywania podstawowych działań arytmetycznych: dodawania, odejmowania, mnożenia, dzielenia, obliczania pochodnej i całkowania;
— możliwość wykorzystania wielomianów do aproksymacji funkcji ciągłych z dowoiną dokładnością.

Wykonanie takich podstawowych działań arytmetycznych, jak dodawanie i odejmowanie wielomianów, z reguły nie sprawia nam trudności, gdyż sprowadza się do redukcji wyrazów podobnych. Natomiast mnożenie i dzielenie wielomianów wymaga nie tylko znajomości odpowiednich metod, ale cierpliwości w wykonywaniu żmudnych obliczeń. W takich przypadkach okazuje się pomocny mikrokomputer. Zamieszczony poniżej program WIELOMIANY wspomaga wykonanie tych operacji. Jest on tak skonstruowany, że pozwala w prosty sposób rozbudować swe możliwości o dodatkowe operacje na wielomianach.

Program WIELOMIANY opracowany został na mikrokomputery: Commodore 64, Plus/4 i 16. Dla mikrokomputerów Commodore 16 i Plus/4 należy wiersz 10 zmienić według wzoru (patrz 10 str. 26).

Program WIELOMIANY umożliwia wykonanie działań (mnożenia i dzielenia) na wielomianach postaci:

$$A(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0$$
$$B(x) = b_m x^m + b_{m-1}x^{m-1} + \ldots + b_0$$

gdzie $a_n \neq 0$; $b_m \neq 0$.

Zapisując wielomiany (np. $A(x) = 2x^4 + x^3 + x - 1$) podajemy:

— stopień wielomianu (w naszym przykładzie jest równy 4)
— dla współczynników różnych od zera: numer współczynnika i jego wartość (nr 0, wartość —1; nr 3, wartość 1; nr 1, wartość 1; nr 4, wartość 2).

Zapisywanie wielomianu kończy się po podaniu wartości współczynnika o numerze równym stopniowi wielomianu. Istnieje możliwość modyfikacji postaci wielomianów przy ponownym wykorzystaniu operacji zapisu.

W wyniku operacji mnożenia wielomianów A(x) i B(x) uzyskujemy wielomian $W(x) = A(x) \cdot B(x)$. W wyniku operacji dzielenia wielomianu A(x) przez wielomian B(x) otrzymamy iloraz W(x) i resztę z dzielenia R(x) spełniające równość:

$$A(x) = W(x) \cdot B(x) + R(x)$$

Wyniki wyprowadzane są według konwencji zapisu, tzn. numery i wartości współczynników różnych od zera.

Algorytmy mnożenia i dzielenia wielomianów wykorzystywane w programie WIELOMIANY są odmienne od tych, jakimi posługujemy się w obliczeniach ręcznych. Zainteresowanych namawiam do prześledzenia tych algorytmów i na ich podstawie opracowania szczególnych algorytmów dzielenia wielomianu przez dwumian $x - b_0$ lub trójmian kwadratowy $x^2 + b_1x + b_0$.

Dla osób, które będą chciały rozbudować program WIELOMIANY, podaję krótki opis kodu programu:

1. wiersze 20—70: wyświetlenie informacji o operacjach aktualnie możliwych do realizacji; wybór operacji;
   wiersze 90—160: zapis wielomianu A(x)
   wiersze 170—230: zapis wielomianu B(x)
   wiersze 240—300: wykonanie dzielenia wielomianów A(x) przez B(x)
   wiersze 310—350: wykonanie mnożenia wielomianów
   wiersze 360—560: wyprowadzenie wyników w zależności od ostatnio wykonanej operacji
   wiersze 600—620: „czyszczenie" ekranu po obejrzeniu partii wyników.
2. zmienne sterujące:
   T — określa aktualnie dostępne operacje;
   D — określa matrycę, według której należy wyprowadzać ostatnio otrzymane wyniki.

Uwaga: w tekście programu zamiast znaku $ użyty jest znak ×

```
0 REM ***WIELOMIANY***
10 T=1:DIMW(39):C=1064:POKE53280,0:
POKE53281,0
20 PRINTCHR×(147)"OPCJE:":ONTGOTO60
,50,40,30
30 PRINT"6 - WYNIKI"
40 PRINT"5 - MNOZENIE WIELOMIANOW A
(X)*B(X)":PRINT"4 - DZIELENIE WIELO
MIANOW A(X)/B(X)"
50 PRINT"3 - ZAPIS WIELOMIANU B(X)"
60 PRINT"2 - ZAPIS WIELOMIANU A(X)"
:PRINT"1 - KONIEC":PRINT:INPUT"WYBI
ERAM";O%
70 PRINTCHR×(147):ON O% GOSUB 80,90
,170,240,310,360:GOTO20
80 END
90 IFT>1THEN120
100 INPUT"MAKSYMALNY STOPIEN WIELOM
IANOW";MS%:IFMS%<0THEN100
110 DIMA(MS%),B(MS%),C(MS%+MS%):T=2
120 INPUT"AKTUALNY STOPIEN WIEL. A(
X)";SA%:IFSA%>MS%ORSA%<1THEN120
130 INPUT"A(X):  NR WSP.";NW%:IFNW%
>SA%ORNW%<0THEN130
140 INPUT"A(X):  WARTOSC";A(SA%-NW%
):IFNW%<>SA%THEN130
150 IFA(0)=0THEN140
160 RETURN
170 IFT<2THENRETURN
```

dokończenie na str. 26

dokończenie ze str. 25

```
180 INPUT"AKTUALNY STOPIEN WIEL. B(
X)";SB%:IFSB%>MS%ORSB%<1THEN180
190 INPUT"B(X):  NR WSP.";NW%:IFNW%
>SB%ORNW%<0THEN190
200 INPUT"B(X):  WARTOSC";B(SB%-NW%
):IFNW%<>SB%THEN190
210 IFB(0)=0THEN200
220 IFT=2THENT=3
230 RETURN
240 IFT<3THENRETURN
250 IFSB%>SA%THENFORK=0TOSA%:C(K)=A
(K):NEXT:D=2:RETURN
260 S=SA%-SB%:FORK=0TOSA%:JP=K-SB%:
IFJP<0THENJP=0
265 JK=K-1:IFK>STHENJK=S
270 C(K)=0:FORJ=JPTOJK:C(K)=C(K)+C(
J)*B(K-J):NEXT:C(K)=A(K)-C(K)
280 IFK<=STHENC(K)=C(K)/B(0)
290 NEXT:IFT=3THENT=4
300 D=1:RETURN
310 IFT<3THENRETURN
320 FORK=0TOSA%+SB%:JP=K-SA%:IFJP<0
THENJP=0
330 JK=SB%:IFJK>KTHENJK=K
340 C(K)=0:FORJ=JPTOJK:C(K)=C(K)+B(
J)*A(K-J):NEXT:NEXT:IFT=3THENT=4
350 D=3:RETURN
360 IFT<4THENRETURN
370 K=0:ONDGOSUB380,470,520:PRINT:I
NPUT"OBEJRZALES";Z$:RETURN

380 PRINT"WSPOLCZYNNIKI ILORAZU ROZ
NE OD ZERA:":PRINT:W=1
390 IFC(K)=0THEN410
400 GOSUB600:PRINT"WSP. NR"S-K;TAB(
12)"WARTOSC:"C(K)
410 K=K+1:IFK<=SA%-SB%THEN390
415 PRINT:INPUT"OBEJRZALES";Z$:PRIN
TCHR$(147)
420 PRINT"WSPOLCZYNNIKI RESZTY ROZN
E OD ZERA:":PRINT:W=1
430 IFC(K)=0THEN450
440 GOSUB600:PRINT"WSP. NR:"SA%-K;T
AB(12)"WARTOSC:"C(K)
450 K=K+1:IFK<=SA%THEN430
460 RETURN
470 PRINT"WSPOLCZYNNIKI RESZTY ROZN
E OD ZERA:":PRINT:W=1
480 IFC(K)=0THEN500
490 GOSUB600:PRINT"WSP. NR"SA%-K;TA
B(12)"WARTOSC:"C(K)
500 K=K+1:IFK<=SA%THEN480
510 RETURN
520 PRINT"WSPOLCZYNNIKI ILOCZYNU RO
ZNE OD ZERA:":PRINT:W=1
530 IFC(K)=0THEN550
540 GOSUB600:PRINT"WSP. NR:"SA%+SB%
-K;TAB(12)"WARTOSC:"C(K)
550 K=K+1:IFK<=SA%+SB%THEN530
560 RETURN
600 W=W+1:IFW<20THENRETURN
610 W=0:PRINT:INPUT"OBEJRZALES";Z$:
FORI=0TO39:W(I)=PEEK(C+I):NEXT
620 PRINTCHR$(147):FORI=0TO39:POKEC
+I,W(I):NEXT:PRINT:PRINT:RETURN


10 T=1:DIMW(39):C=3112:COLOR0,1,1:
COLOR4,1,1
```

PROGRAM ATARI 32

Program może służyć do nauki alfabetu Morse'a z dźwiękowym i wizualnym jego odwzorowaniem lub po podłączeniu mikrokomputera do nadajnika jako moduł usprawniający pracę operatora.

Korzystając z tego programu można nadawać klawiaturą lub kluczem. W obu przypadkach jest możliwe otrzymanie na ekranie znaków Morse'a lub alfabetu łacińskiego. Ponadto takie kody jak: nazwa stacji, wywołanie ogólne itp., nadawane są w całości przez naciśnięcie określonego klawisza. W tym celu należy powrócić do MENU wciskając klawisz „ * ", a następnie wcisnąć klawisz z literą występującą przy danym kodzie.

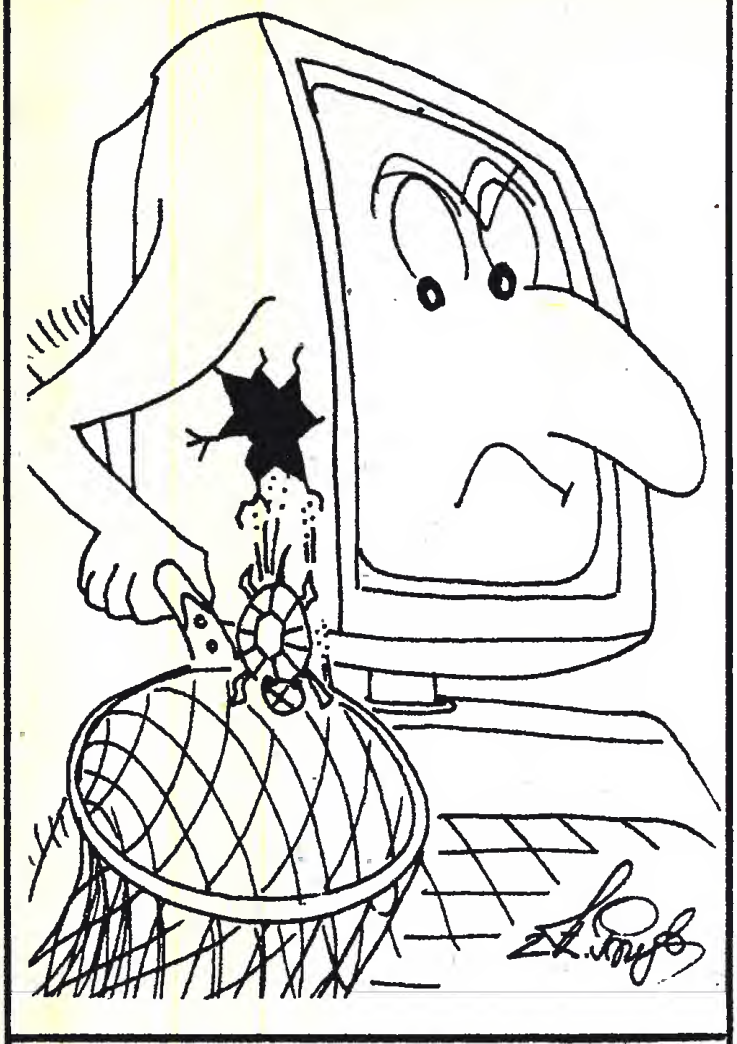
W miejsce nazwy stacji SP5IVJ należy wpisać swoją nazwę. Dokonuje się tego w L.136 i L.4000 (na końcu ciągu należy pozostawić gwiazdkę). Gdyby użytkownik zechciał rozszerzyć zestaw znaków alfabetu Morse'a, wystarczy wprowadzić do programu dane pomiędzy L.20050 i L.20060 np. 20055 DATA (znak), (kod znaku).

Szybkość emitowania znaków za pomocą klucza zależy od sprawności manualnej użytkownika oraz jakości klucza. Stąd też w przypadku znaków dekodowanych i otrzymywanych w postaci alfabetu łacińskiego na ekranie monitora wprowadzono pętlę L.2120,2140. W pętli tej zmienną CZ ustala się czas, w jakim jest dekodowany wprowadzony znak. Zmienna CZ obliczana jest w L.2103 na podstawie danej Z wprowadzonej w L.2102.

Jako klucz nadaje się najbardziej przedstawiony na rysunku manipulator klucza elektronowego. Przyłącza się go do wyprowadzeń nr 3, 4 i 8 (GND) gniazda joystick'a oznaczonego na obudowie mikrokomputera jako CONTRLLERS 1. Zamiast klucza można stosować również joystick, jednak powoduje to znacznie wolniejsze nadawanie. Przekazywanie sygnału z mikrokomputera do nadajnika dokonuje się z końcówek nr 3 i 2 (GND) gniazda MONITOR.

```
5 REM **Alfabet Morse'a**
6 REM (C) Janusz J.
7 REM Warszawa 1986,7
10 OPEN #1,4,0,"K:"
12 ? "}":POKE 752,1
14 GRAPHICS 2
16 POSITION 2,1:? #6;"-.-.-.-.-.-.-.-.-"
18 POSITION 2,3:? #6;"alfabet morse'a"
20 POSITION 2,5:? #6;"(C)      1986,7"
22 POSITION 2,7:? #6;"-.-.-.-.-.-.-.-.-"
25 DIM B(12,9),SP$(7),L$(10)
26 DIM WL$(1),DL$(1),A$(6)
28 DIM KL$(6),AL$(1),K$(6)
29 L$="0123456789"
30 FOR E=0 TO 4
35 FOR F=0 TO 4
40 EF=E+F:FE=INT(0.3*F*E)
45 B(F,EF)=45
50 B(F+6,EF+1)=46
55 B(F+6,FE)=45
60 B(F+1,FE)=46
65 NEXT F:NEXT E
66 B(10,0)=45:B(10,1)=46:B(10,2)=46
B(10,3)=46:B(10,4)=45
67 B(11,0)=45:B(11,1)=46:B(11,2)=46
B(11,3)=45:B(11,4)=46
70 ? :? "     Wcisnij dowolny klawisz"
75 GET #1,P
80 GRAPHICS 0+16
82 POKE 752,1
84 POKE 764,147
100 ? "}"
110 POSITION 16,5:? " MENU "
120 POSITION 7,7:? "A - KORZYSTANIE
Z KLAWIATURY"
130 POSITION 7,8:? "B - KORZYSTANIE
Z KLUCZA"
135 POSITION 15,11:? "  KODY: "
136 POSITION 14,13:? "D - SP5IVJ"
137 POSITION 14,14:? "E - QRZ"
138 POSITION 14,15:? "F - QRA"
139 POSITION 14,16:? "G - QRT"
140 POSITION 14,17:? "H - QRX"
141 POSITION 14,18:? "I - QTH"
160 GET #1,AM
165 IF AM<65 OR AM>73 THEN 160
170 MA=AM-64
180 GOTO 1000*MA
1000 REM WYKORZYSTANIE KLAWIATURY
1010 GOSUB 1500
1020 ? "}"
1025 IF AK=42 THEN 100

1030 GET #1,X
1032 IF X=42 THEN 1010
1033 IF AK=65 THEN ? CHR$(X);
1035 IF X>46 AND X<58 OR X=61 THEN 1400
1040 WL$=CHR$(X)
1050 RESTORE
1060 READ DL$,A$
1065 L=LEN(A$)
1070 IF DL$="$" THEN ?"BRAK DANYCH #
";:GOTO 1030
1080 IF DL$=WL$ THEN 1200
1090 GOTO 1060
1200 FOR D=1 TO L
1210 IF AK=65 THEN 1230
1220 ? A$(D,D);
1230 IF A$(D,D)="-" THEN GOSUB 1300
:GOTO 1245
1240 FOR F=1 TO 8:SOUND 0,20,14,15:
NEXT F
1242 FOR F=1 TO 10:NEXT F
1245 SOUND 0,0,0,0
1250 NEXT D
1255 ? " ";
1260 GOTO 1030
1300 FOR F=1 TO 30
1310 SOUND 0,20,14,15
1320 NEXT F
1330 RETURN
1400 Z=X-48
1402 IF Z=13 THEN Z=10
1404 IF Z=-1 THEN Z=11
1410 FOR FF=0 TO 4
1415 IF AK=65 THEN 1425
1420 ? CHR$(B(Z,FF));
1425 IF B(Z,FF)=45 THEN GOSUB 1300:
GOTO 1432
1430 FOR U=1 TO 8:SOUND 0,20,14,15:
NEXT U
1431 FOR U=1 TO 10:NEXT U
1432 SOUND 0,0,0,0
1435 NEXT FF
1440 ? " ";
1450 GOTO 1030
1500 ? "}"
POSITON 7,5:? "CO CHCESZ NA
EKRANIE ?"
1520 POSITION 10,7:? "A - alfabet"
1530 POSITION 10,8:? "B - kod Mors'
1540 POSITION 10,10:? "* - powrot
 do menu."

1550 GET #1,AK
1560 RETURN
2000 REM WYKORZYSTANIE KLUCZA
2004 POKE 764,255
2005 GOSUB 1500
2007 IF AK=65 THEN 2100
2008 IF AK=42 THEN 100
2010 ? "}"
2015 ST=STICK(0)
2020 IF ST=11 THEN ? ".";:FOR F=1 T
O 8:SOUND 0,20,14,15:NEXT F
2030 IF ST=7 THEN ? "-";:FOR F=1 TO
30:SOUND 0,20,14,15:NEXT F
2040 SOUND 0,0,0,0
2050 IF PEEK(764)=7 THEN 2004
2060 GOTO 2015
2100 POSITION 10,14:? "CZAS DEKODOW
ANIA ? "
2101 POSITION 17,15:? "(10 - 60)"
2102 POSITION 27,14:? ;:INPUT Z
2103 CZ=INT(120/Z)
2105 ? "}"
2110 DK=1
2115 KL$=""
2120 FOR R=1 TO CZ
2121 SOUND 0,0,0,0
2123 ST=STICK(0)
2125 IF ST=11 THEN 2210
2130 IF ST=7 THEN 2230
2133 L=LEN(KL$)
2134 IF L=6 THEN 2175
2135 IF PEEK(764)=7 THEN 2004
2136 IF L=5 THEN 2300
2140 NEXT R
2170 IF L=0 THEN 2120
2175 RESTORE
2180 READ AL$,K$
2185 IF AL$="$" THEN 2400
2190 IF K$=KL$ THEN ? AL$;:GOTO 2110
2195 GOTO 2180
2210 IF DK>6 THEN 2400
2212 KL$(DK,DK)="."
2215 FOR F=1 TO 8:SOUND 0,20,14,15
:NEXT F
2220 DK=DK+1
2225 GOTO 2120
2230 IF DK>6 THEN 2400
2232 KL$(DK,DK)="-"
2235 FOR F=1 TO 30:SOUND 0,20,14,15
:NEXT F
2240 DK=DK+1
2245 GOTO 2120
2300 FOR G=0 TO 11
2310 FOR E=0 TO 4
2315 N=E+1
```

dokończenie na str. 28



```
2317 IF G=11 AND ASC(KL$(N,N))<>B(11
,E) THEN 2400
2320 IF ASC(KL$(N,N))<>B(G,E)THEN 2
330
2323 IF E=4 AND ASC(KL$(5,5))=B(G,4
) THEN 2333
2325 NEXT E
2330 NEXT G
2333 IF G=10 THEN ? "=";:GOTO 2110
2335 IF G=11 THEN ? "/";:GOTO 2110
2350 ? G;
2360 GOTO 2110
2400 ? "BRAK OKRESLENIA ZNAKU";
2410 GOTO 2110
4000 SP$="SP5IVJ*"
4005 LS=LEN(SP$)
4010 GOSUB 4500
4020 GOTO 160
4500 N=0
4510 N=N+1
4515 IF LS>4 AND N=3 THEN 4800
4520 RESTORE
4530 READ DL$,A$
4540 IF A$="*" THEN RETURN
4550 IF DL$=SP$(N,N) THEN 4570
4560 GOTO 4530
4570 L=LEN(A$)
4580 FOR F=1 TO L
4590 IF A$(F,F)="." THEN GOSUB 4930
:GOTO 4690
4610 GOSUB 4900
4690 NEXT F
4700 FOR E=1 TO 12
4710 NEXT E
4720 GOTO 4510
4800 FOR M=1 TO 10
```

```
4810 IF L$(M,M)=SP$(N,N) THEN 4820
4815 NEXT M
4820 MM=M-1
4830 FOR F=0 TO 4
4840 IF B(MM,F)=46 THEN GOSUB 4930
:GOTO 4860
4850 GOSUB 4900
4860 NEXT F
4890 GOTO 4510
4900 FOR E=1 TO 19
4910 SOUND 0,20,14,15
4920 NEXT E
4930 FOR E=1 TO 5
4940 SOUND 0,20,14,15
4950 NEXT E
4960 SOUND 0,0,0,0
4970 FOR E=1 TO 5
4975 NEXT E
4980 RETURN

5000 SP$="QRZ*"
5010 GOSUB 4500
5020 GOTO 160
6000 SP$="QRA*"
6010 GOSUB 4500
6020 GOTO 160
7000 SP$="QRT*"
7010 GOSUB 4500
7020 GOTO 160
8000 SP$="QRX*"
8010 GOSUB 4500
8020 GOTO 160
9000 SP$="QTH*"
9010 GOSUB 4500
9020 GOTO 160
20000 DATA A,.-,B,-...,S,...,T,-,G,--.
20010 DATA J,.---,N,-.,O,---,K,-.-,Y,-.--
20020 DATA M,--,F,..-.,Z,--..,I,..,X,-..-
20030 DATA D,-..,R,.-.,H,....,E,.,W,.--
20040 DATA L,.-..,Q,--.-,P,.--.,V,...-,C,-.-.
20050 DATA U,..-,?,..--..,!,-.-.--
20060 DATA $,*
```

Wideo — straszak

Cokolwiek zrobilibyśmy — będzie źle. Nie było w Polsce komputerów – źle. Są — też źle. Taka ocena sytuacji przypomina starą anegdotę. Znacie? No to, posłuchajcie! Otóż w knajpce siedzi kilku „podciętych" facetów. Nudzi im się. Spostrzegają przy sąsiednim stoliku chudego, bladego faceta w okularach. Rozmowa „zmęczonych" nabiera rozmachu.

— Trzeba by mu dać w ucho!

— Ale za co?

— Spytaj się go czy był kiedyś v. Gródku...

— A jak był?

— Dasz mu w ucho.

— A jak nie był?

— Też dasz mu w ucho!

Ludzie nam niechętni zawsze znajdą powód, aby próbować wbić przysłowiową szpilę. Tym razem okazją stał się minikomputery i videokasety. Niedawno londyński dziennik „The Times" opublikował artykuł pod frapującym tytułem: *„Początek innych wojen gwiezdnych"*. Autor, mister Roger Boyes rozwodził się długo i boleśnie na temat zagrożenia, jakie rzekomo dla Polski i w ogóle krajów socjalistycznych stanowią owe minikomputery videokasety. Dlaczego?

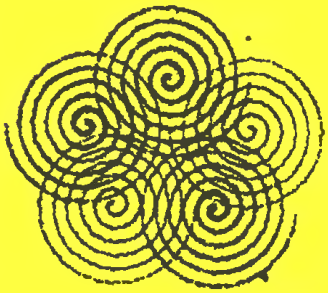
Ponieważ można je wykorzystać do antypaństwowej roboty, np. w zaciszu domowym, wyświetlać dla rodziny, a co gorsze dla sąsiadów, tanp. film jak „Amadeusz" Miłosa Formana, który z pewnością uważany jest „tam", to znaczy — u nas, za „niebezpieczny ładunek". I właśnie, dlatego, mieliśmy okazję zobaczyć go przed paroma miesiącami na „Konfrontacjach'86".

Henryk KAWKA

Giełda... październik'86
Foto: J. Zelman

JANUSZ MILLER
SHARP PC-1500
S P I R A L A

```
10:BEEP 10,100,10
   0
20:"A":CLEAR :
   USING :WAIT 0:
   RADIAN :GRAPH
30:INPUT "ILE SPI
   RALI ? N=";N:N
   =ABS N:N=INT N
   :IF N=0GOTO 10
40:INPUT "ILE ZWI
   TEK ? Z=";Z:Z=
   ABS Z:Z=INT Z:
   IF Z=0GOTO 10
50:A=2*π/N:DIM W(
   N,2)
60:S=π/40:Q=1/(Z*
   2):PRINT "N";N
   ;" Z";Z
70:FOR K=0TO (N-1
   )
80:M=M+1:W(M,1)=6
   0*COS (K*A)
90:W(M,2)=50*SIN
   (K*A)
100:NEXT K
110:RLINE -(80,0),
   9:SORGN
120:FOR K=1TO N
130:X=W(K,1):Y=W(K
   ,2)
140:LINE -(0,0)-(X
   ,Y),9:SORGN
150:T=0:R=0:X2=0:Y
   2=0:U=U+1:IF U
   =4LET U=1
160:COLOR U
170:FOR J=1TO Z
180:FOR L=1TO 80
190:X1=X2:Y1=Y2:T=
   T+S:R=R+Q:X2=R
   *COS T
200:Y2=R*SIN T:IF
   (K-INT (K/2)*2
   )=0LET X2=-X2
210:LINE -(X1,Y1)-
   (X2,Y2),1
220:NEXT L:NEXT J
230:LINE -(X2,Y2)-
   (0,0),9
240:NEXT K:TEXT
250:END
```



LIGA MYŚLĄCYCH

Zadanie nr 1

Na drodze o długości 720 m przednie koło wozu wykonało o 96 obrotów więcej niż tylne. Gdyby obwód każdego koła był o pół milimetra mniejszy, to przednie koło na tej samej drodze wykonałoby o 128 obrotów więcej niż tylne.

Obliczyć długość obwodu każdego koła.

Zadanie nr 2

W szkole przeprowadzono ankietę wśród uczniów na temat nowo wprowadzonego programu nauczania z matematyki. Jeden z uczniów, miłośnik łamigłówek w rubryce wiek ankietowanego wpisał: jeśli mój wiek, który będę miał za trzy lata, trzykrotnie zwiększycie i odejmiecie od tego powiększony trzykrotnie mój wiek, który miałem trzy lata temu, to dowiecie się ile mam obecnie lat.

Zadanie nr 3

Na przebycie drogi o długości 520 km, jeden podróżny zużyje o 3 dni więcej czasu niż drugi, który przebywa dziennie o 12 km więcej niż pierwszy. Ile dni potrzebuje każdy z nich na przebycie tej drogi?

Zadanie nr 4

W trójwymiarowej przestrzeni danych jest n-punktów, z których żadne cztery nie leżą na jednej płaszczyźnie. Ile prostych i ile płaszczyzn wyznaczają te punkty?

Rozwiązania zadań prosimy przesyłać do redakcji do końca listopada br., z dopiskiem „Liga Myślących". Punktacja zależy od liczby prawidłowych rozwiązań. Wśród uczestników rozlosujemy książki, a na zwycięzców „Ligi" czekają dodatkowe cenne nagrody — niespodzianki.

Odpowiedzi do zadań z 4 nr. „IKS-a"

Zadanie 1

Z warunków zadania wiemy, że z każdą godziną różnica wskazań zegarów rośnie o 3 minuty. Zatem, od momentu wyregulowania i nakręcenia, 1 godzinę różnicy wskazań, zegary osiągnęły po 20 godzinach. Stąd, do chwili zatrzymania, budzik pośpieszył się o 20 minut. Z tego wynika, że w momencie zatrzymania zegarów była godzina 7.40 (od chwili, w której stanął budzik tj. od godziny 8.00 odejmujemy 20 minut). Wyregulowanie i nakręcenie zegarów odbyło się 20 godzin wcześniej, tj. o godzinie 11.40 dnia poprzedniego.

Zadanie 2

Wszystkich możliwości wylosowania trzech różnych cyfr z dziesięciu, a więc wszystkich kombinacji 3-elementowych z 10 jest:

$$\binom{3}{10} = \frac{10!}{3!\,(10-3)!}, \text{ tj. } 120.$$

Sprzyjających kombinacji czyli takich, kiedy suma cyfr równa się 9 lub 18 jest tylko 14 (018, 027, 036, 045, 126, 135, 234, 378, 468, 567, 918, 927, 936, 945). A więc prawdopodobieństwo wyciągnięcia „na chybił trafił" spośród 10 cyfr od 0 do 9 trzech takich, aby można z nich ułożyć liczbę podzielną przez 9 wynosi:

$$P = \frac{14}{120} = \frac{7}{60}$$

Zadanie 3

Jeżeli n-2 zawodników rozegrało między sobą partię każdy z każdym, to rozegrano tyle partii ile jest 2-elementowych kombinacji ze zbioru n-2-elementowego:

$$\binom{n-2}{2} = \frac{(n-3)(n-2)}{2}$$

Gdy założymy, że pozostali dwaj gracze nie rozegrali partii między sobą, to z warunków zadania wynika, że:

$$\frac{(n-3)(n-2)}{2} + 10 + 1 = 55,$$

czyli $n^2 - 5n - 82 = 0$.

Równanie to nie posiada rozwiązań w zbiorze liczb naturalnych a zatem założenie, że dwaj gracze nie rozegrali między sobą partii było błędne.

Załóżmy więc, że pozostali dwaj gracze rozegrali między sobą partię. Wtedy otrzymamy następujące równanie:

$$\frac{(n-3)(n-2)}{2} + 10 = 55$$

czyli $n^2 - 5n - 84 = 0$.

Rozwiązaniem tego równania są $n_1 = 7$, które odrzucamy i $n_2 = 12$.

Zatem w turnieju szachowym brało udział 12 zawodników, a wspomniani dwaj gracze rozegrali między sobą partię.

NAZWISKA OSÓB, które wylosowały książki za poprawne rozwiązanie zadań z 4 nr. „IKS-a".

1. Krzysztof PAROL z Mińska Mazowieckiego (05-300 ul. Obrońców Stalingradu 27/11)
2. Bartosz BOBER z Rzeszowa (35-106 ul. Wnukowskiego 47/44)
3. Witold ŻYŁA z Warszawy (02-123 ul. Korotyńskiego 19a/32)
4. Beata BIESZK ze Szczecina (70-136, ul. 9 Maja 8/21

Pocztowa giełda

Rozwiązanie krzyżówki z „IKS-a" nr 4

Hasło brzmi: „Z komputerem do szkoły". Bony pieniężne (1000 zł) wylosowali: Maciej Janik — Częstochowa, Grażyna Kolbusz — Brzeg, Adam Kardasz — Gdańsk, Jarosław Płąstka — Lucynowo/Ostrowite, Sławomir Rydel — Kolno.

Nagrody książkowe otrzymują: Monika Andrzejewska — Ostrołęka, Robert Kalisz — Wrocław, Tomasz Łapiński — Pruszków, Michał Sułkowski — Tarnobrzeg, Dariusz Domański — Żarów, Adela Kopyciok — Świętochłowice, Sylwester Bąk — Lublin, Mirosława Orłowska — Kłodzko, Beata Bobowska — Wrocław, Barbara Siarkiewicz — Sosnowiec.

UWAGA — KONKURS!

Miłośnicy literatury science fiction z oryginalnymi pomysłami i fantazją! Każdy z Was ma szansę spróbować swoich sił w naszym konkursie na nowelę lub opowiadanie pod hasłem „Przed nami XXI wiek".

„Najfantastyczniejsze" utwory zostaną uhonorowane przez ziemskie jury, które przyzna nagrody:

I — 15 tys. złotych
II — 10 tys. złotych
III — 5 tys. złotych

Nad ich sprawiedliwym i obiektywnym podziałem czuwać będzie komputer. Autorzy materiałów zakwalifikowanych do druku zostaną ponadto usatysfakcjonowani godziwym honorarium zgodnie z obowiązującymi stawkami.

Entuzjastów informatyki i komputerów nie musimy przekonywać, że i fantazja musi mieć swoje ścisłe parametry. A więc mile widziane będą prace o objętości nie przekraczającej 15 stron maszynopisu bądź czytelnego rękopisu.

Jak będzie wyglądał nasz glob opanowany przez komputery n-tej generacji, w jaki sposób informatyka zmieni życie człowieka XXI wieku, jak homo sapiens ułoży sobie koegzystencję z rozumnymi maszynami? To tylko niektóre sugestie organizatorów dotyczące tematyki nadsyłanych utworów.

Prace opatrzone godłem należy nadsyłać do końca grudnia, dołączając kopertę podpisaną godłem, zawierającą dane o autorze, pod adresem:

„IKS" — ul. Grzybowska 77
00-950 W-wa

Przychylności weny, zdyscyplinowanej fantazji — życzy **REDAKCJA**

PS Zastrzegamy sobie prawo pierwszeństwa publikacji prac nagrodzonych oraz wybranych spośród nie nagrodzonych.

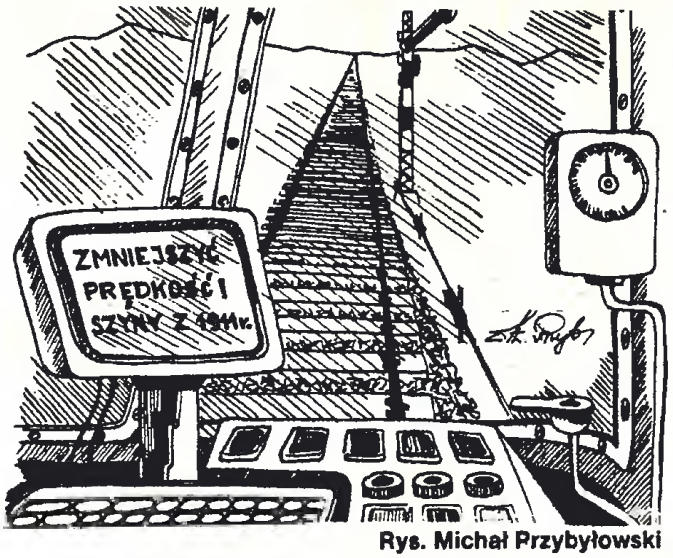
W naszym komputerlandzie

W zakresie kształcenia intelektualnego pracowity Spektruś „zabezpieczył" dla kierowanego przez siebie zespołu robotów uznanego lektora. Przygotował on wykład oraz dyskusję panelową na temat: „Dlaczego w komputerlandzie jest tyle znaków nierówności, a tak mało równania do przodu?".

Zaplanowano trzy spotkania, w których wziąć miały udział wszystkie roboty a także sam Spektruś. Prelegent jednak najzwyklej przynudzał i po dwudziestu minutach pierwszego wykładu okazało się, że połowa miejsc na sali jest pusta bo niesforne roboty wycofały się po angielsku, żeby nie zwracać uwagi. W połowie drugiego spotkania wyłączono światło i zanim znaleziono świeczki lektor oberwał po głowie i mógł jedynie bredzić. Tym samym prowadził wykład na takim samym poziomie, jak w pierwszej części. Do trzeciego spotkania nie doszło, bo wcześniej ktoś zamknął lektora w ustronnym miejscu i zgubił klucz od skobelka. Kiedy go wyswobodzono sala była już pusta.

Aby uniknąć takich sytuacji w przyszłości i zapewnić wysoki poziom prelekcji zapobiegliwy Spektruś opracował kompetentną i wszechstronną ściągę dla prelegentów na każdą okazję. Jego zdaniem najlepsze efekty można uzyskać jeśli jedną trzecią odczytu stanowią stare, odwieczne prawdy, żeby słuchacze doszli do wniosku, że wcale nie są głupi — a przeciwnie. Środkową część należy poświęcić zagadnieniom nowym lecz podanym w przystępnej formie, by słuchaczom zdało się, iż z odczytu odnoszą korzyść. Ostatnia część powinna traktować o takich czy innych zjawiskach językiem niezrozumiałym, aby słuchacze nie stracili szacunku dla nauki. No i osoby prelegenta...

podglądał: Eugeniusz MLECZAK



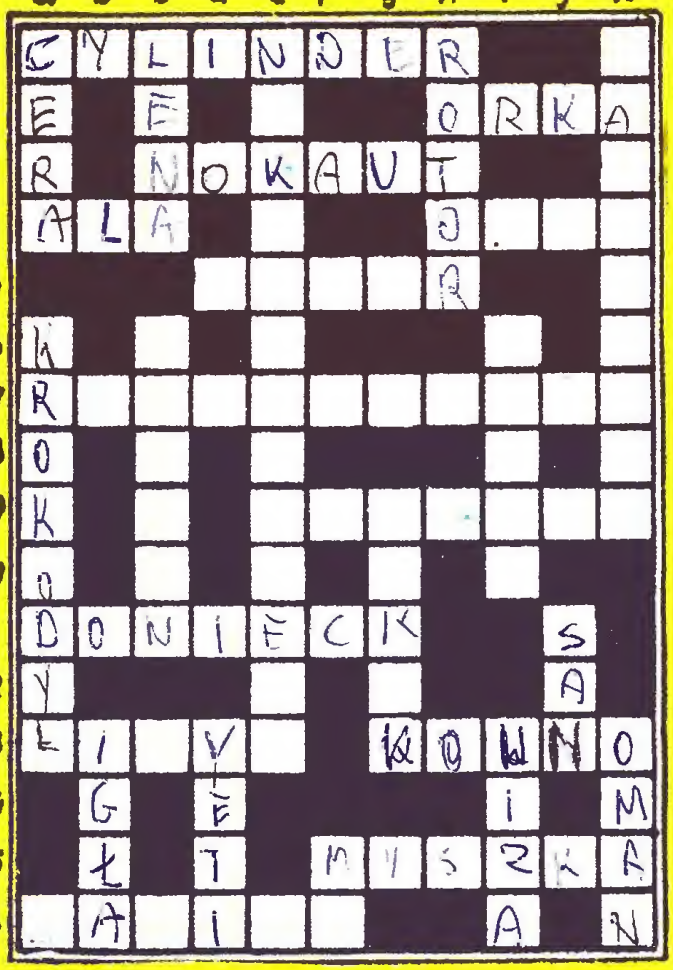
Rys. Michał Przybyłowski

Przepraszamy...

„IKS" nr 5, program „Wykres funkcji" — powtarzamy nieczytelne fragmenty

```
500 REM WYZNACZENIE WSP. SKALI
510 XZ=XK-XP:YZ=YG-YD
520 DX=XZ/239:DY=YZ/159
530 CY=INT(ABS(YD*(YD<0))/DY):CY=160-CY
540 CX=INT(ABS(XP*(XP<0))/DX)+40
```

Krzyżówka nr 6

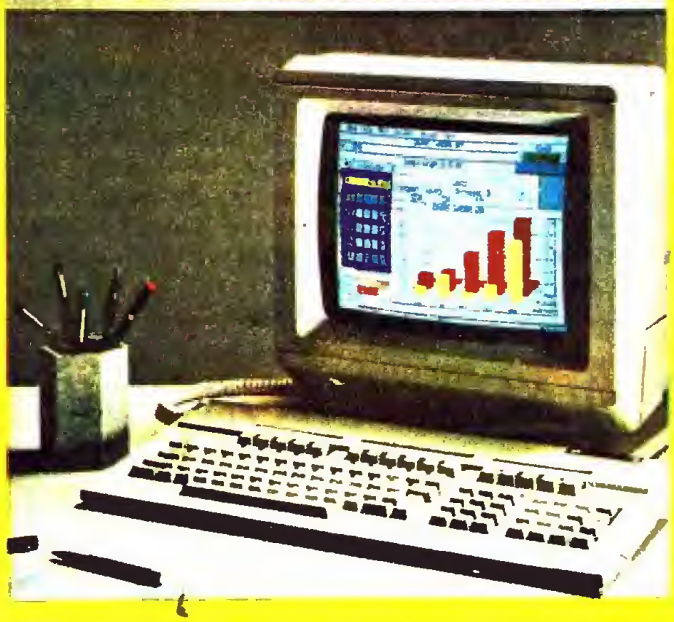


POZIOMO: 1-a) Kapelusz lub część silnika, 2-h) Praca w polu, 3-c) „Wypadek" na ringu, 4-a) Koleżanka Oli, 4-h) Jeden z gromady stawonogów, 5-d) Tkanina runowa lub miasto w Indiach, 7-a) Światłoczuły barwnik w pręcikach siatkówki oka, 9-e) Rodzaj kaktusa, 11-a) Miasto w ZSRR, 13-a) Rój meteorów, 13-g) Port nad Niemnem, 15-f) Gryzoń z komputera, 16-a) Wygnaniec.

PIONOWO: 1-a) Ważna dla kobiety, 1-c) Rzeka lub imię żeńskie, 1-e) Występuje w jądrze i cytoplazmie każdej komórki, 1-h) Wirnik, 1-k) Kapłan w przedchrześcijańskiej Litwie, 6-a) Mieszkaniec Nilu, 6-c) Imię męskie, 6-l) W piosence nauczycielka..., 9-g) Deformacja tektoniczna warstw skalnych, 11-j) Autobus lub rzeka, 13-b) Tańcowała z nitką, 13-d) Tajemnica Tybetu, 13-i) Zezwolenie na pobyt za granicą, 13-k) Państwo na Płw. Arabskim, 15-f) Jedna ze stron konta.

Hasło: 9-f, 7-a, 15-i, 13-d, 9-f, 2-h, 15-f, 9-c, 13-j, 2-k, 15-f, 13-d, 3-d, 9-f, 1-h, 5-e, 9-h, 5-g, 14-k, 1-g, 5-h, 12-j, 9-l, 13-b, 14-d, 16-d, 11-g, 15-h, 4-c.

Rozwiązania (tylko hasło) prosimy przesyłać pod adresem redakcji do końca listopada. Wśród Czytelników, którzy nadeślą prawidłowe rozwiązania rozlosujemy bony pieniężne i nagrody książkowe.



Oto najnowszy produkt firmy ICL — DRS 300. Najważniejszą jego zaletą jest modułowa konstrukcja; podstawową konfigurację można łatwo rozbudowywać dołączając kolejne urządzenia. Nie ma przy tym piątaniny kabli; foremne moduły idealnie pasują do siebie, a można je ustawić nawet w podręcznej biblioteczce.

Rys. Michał Przybyłowski

„IKS" — dodatek „Żołnierza Wolności". Redagują: Wiesław Cetera, Ryszard Rogoń. Adres redakcji: 00-950 Warszawa ul. Grzybowska 77, telefon centrali 20-12-61 w. 486. Rękopisów nie zamówionych redakcja nie zwraca i zastrzega sobie prawo do skrótów. Nakładem: Wydawnictwa „Czasopisma Wojskowe", Warszawa ul. Grzybowska 77, Fotoskład i druk rotograwiurowy — Wojskowe Zakłady Graficzne im. gen. dyw. A. Zawadzkiego. Nr zam. 8112. Nr ind. 382809

P-8 5